▼ AMSTRAD ▼ SPECTRUM ▼ COMMODORE ▼ IBM ▼ MSX ▼

Informática y programación 5

PASO A PASO



PROGRAMAS EDUCATIVOS
PROGRAMAS DE UTILIDAD
PROGRAMAS DE GESTION
PROGRAMAS DE JUEGOS

▼ BASIC ▼ MAQUINA ▼ PASCAL ▼ LOGO ▼ OTROS LENGUAJES ▼
▼ TECNICAS DE ANALISIS Y DE PROGRAMACION ▼

▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼

Informática y programación 5

PASO A PASO



PROGRAMAS EDUCATIVOS
PROGRAMAS DE UTILIDAD
PROGRAMAS DE GESTION
PROGRAMAS DE JUEGOS

▼ BASIC ▼ MAQUINA ▼ PASCAL ▼ LOGO ▼ OTROS LENGUAJES ▼
▼ TECNICAS DE ANALISIS Y DE PROGRAMACION ▼

▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼

Una publicación de

EDICIONES SIGLO CULTURAL, S.A.

Director-editor:
RICARDO ESPAÑOL CRESPO.

Gerente:
ANTONIO G. CUERPO.

Directora de producción:
MARIA LUISA SUAREZ PEREZ.

Directores de la colección:
MANUEL ALFONSECA, Doctor Ingeniero de Telecomunicación y Licenciado en Informática.
JOSE ARTECHE, Ingeniero de Telecomunicación.

Diseño y maquetación:
BRAVO-LOFISH.

Fotografía:
EQUIPO GALATA.

Dibujos:
JOSE OCHOA

TECNICAS DE PROGRAMACION: Manuel Alfonseca, Doctor Ingeniero de Telecomunicación y Licenciado en Informática. TECNICAS DE ANALISIS: José Arteche, Ingeniero en Telecomunicación. LENGUAJE MAQUINA 8086: Juan Rojas Licenciado en Ciencias Físicas e Ingeniero Industrial. PASCAL: Juan Ignacio Puyol, Ingeniero Industrial. PROGRAMAS (educativos, de utilidad, de gestión y de juegos): Francisco Morales, Técnico en Informática y colaboradores. Coordinador de AULA DE INFORMATICA APLICADA (AIA): Alejandro Marcos, Licenciado en Ciencias Químicas. BASIC: Esther Maldonado, Diplomada en Arquitectura. INFORMATICA BASICA: Virginia Muñoz, Diplomada en Informática. LENGUAJE MAQUINA Z-80: Joaquín Salvachúa, Diplomado en Telecomuniación y José Luis Tojo, Diplomado en Telecomunicación. LENGUAJE MAQUINA 6502: Jesús Bocho, Licenciado en Informática. LOGO: Cristina Manzanero, Licenciada en Informática. APLICACIONES: Fernando Suero, Diplomado en Telecomunicación. OTROS LENGUAJES (Sistemas operativos): Domingo Villaseñor, Diplomado en Informática, y Lenguaje C: Enrique Serrano, Ingeniero en Telecomunicación.

Ediciones Siglo Cultural, S.A.

Dirección, redacción y administración:
Pedro Texeira, 8, 2.ª planta. Teléf. 810 52 13. 28020 Madrid.

Publicidad:
Gofar Publicidad, S.A. Benito de Castro, 12 bis. 28028 Madrid.

Distribución en España:
COEDIS, S.A. Valencia, 245. Teléf. 215 70 97. 08007 Barcelona.
Delegación en Madrid: Serrano, 165. Teléf. 411 11 48.

Distribución en Ecuador: Muñoz Hnos.

Distribución en Perú: DISELPESA.

Distribución en Chile: Alfa Ltda.

Importador exclusivo Cono Sur:
CADE, S.R.L. Pasaje Sud América, 1532. Teléf.: 21 24 64.
Buenos Aires - 1.290. Argentina.

ISBN del tomo: 84-7688-078-2
ISBN de la obra: 84-7688-068-7

Fotocomposición:
ARTECOMP, S.A. Albarracín, 50. 28037 Madrid.

Imprime:
MATEU CROMO. Pinto (Madrid).

Depósito legal: M. 5.677-1987
Printed in Spain - Impreso en España.

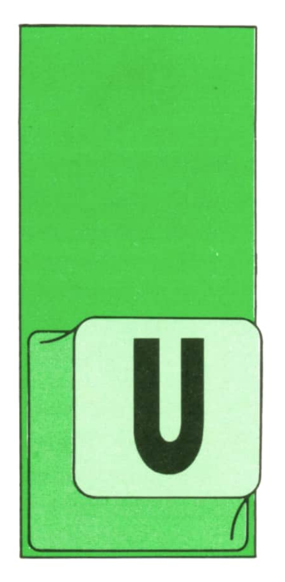
Suscripciones y números atrasados:
Ediciones Siglo Cultural, S.A.
Pedro Texeira, 8, 2.ª planta. Teléf. 810 52 13. 28020 Madrid.

Abril, 1987.
P.V.P. Canarias: 335,–.

INDICE

BASIC

LAS VARIABLES



NA variable es un espacio de la memoria en el que podemos almacenar un dato, numérico o alfanumérico. Este dato se denomina valor de la variable.

Puesto que existen dos tipos de datos, también existen dos tipos de variables:

- Numéricas: sirven para almacenar números.
- Alfanuméricas: sirven para almacenar cadenas de caracteres.

Para distinguir unas variables de otras tenemos que dar un nombre a cada una de ellas.

En general, el nombre de una variable tiene que cumplir los requisitos siguientes:

— La longitud máxima del nombre varía de unos ordenadores a otros, aunque generalmente suele ser bastante grande.

— El primer carácter que forma el nombre de la variable debe ser siempre una letra. El resto pueden ser números y/o letras, pero nunca signos como ?, -, +.

— No se pueden utilizar palabras BASIC como nombres de variables. Algunas máquinas tampoco aceptan nombres de variables que contengan palabras BASIC, como, por ejemplo, ALETA, SENDA.

— Algunos ordenadores sólo distinguen los dos primeros caracteres del nombre; por tanto, si dos nombres de variable distintos comienzan por los dos mismos caracteres, el ordenador interpretará que se trata de la misma variable. Por ejemplo: SUMA, SUELDO, SU28 serían la misma variable.

— El último carácter del nombre puede ser especial ($, !, %, #) y sirve para distinguir distintos tipos de variables, si los hay. En principio, nos basta con saber que el signo $ al final del nombre de una variable la identifica como variable alfanumérica, mientras que si no lleva ningún signo especial como último carácter será una variable numérica.

— Las letras empleadas en un nombre de variable pueden ser mayúsculas o minúsculas, ya que el ordenador no hace ninguna distinción. Por ejemplo, AB, ab, Ab y aB son la misma variable.

En la tabla de la figura 1 podemos ver algunas excepciones a estas normas que presentan los principales ordenadores.

AMSTRAD	Admite como nombre de variable un nombre que contenga una palabra BASIC
COMMODORE	No admite los signos especiales ! y #
IBM	Admite como nombre de variable un nombre que contenga una palabra BASIC. Reconoce todos los caracteres que componen el nombre de la variable
MSX	Ninguna excepción
SPECTRUM	No admite los signos especiales !, % y # Reconoce todos los caracteres que componen el nombre de la variable. Admite como nombre de variable un nombre que contenga una palabra BASIC, o la propia palabra BASIC Sólo admite una letra y el signo $ como nombre de variable alfanumérica

Excepciones a las normas generales sobre variables en los principales ordenadores.

En la tabla de la figura 2 podemos ver el significado de los símbolos %, ! o # al final de una variable numérica.

	Ultimo carácter	Tipo	Característica
Variables Numéricas	%	Enteras	Sólo número enteros
	!	Simple precisión	Aproximadamente 6 ó 7 cifras significativas después del punto decimal
	#	Doble precisión	Aproximadamente el doble de cifras significativas que las de simple precisión

Especificaciones de los diferentes tipos de variables numéricas.

Finalmente podemos ver algunos ejemplos: R, RADIO, TOTALVENTAS, NOMBRE$,

NUM%, C25$, SUM12 son variables.
24B, TO+, $CALLE, A+B, CODIGO-POSTAL no son variables.



ASIGNACION: LET

El proceso mediante el cual damos un valor a una variable recibe el nombre de asignación.

El proceso de asignación cubre los siguientes puntos:

* Reservar un espacio en memoria.
* Dar un nombre a dicho espacio (variable).
* Asignar un valor a la variable.

La asignación se puede realizar de distintas maneras utilizando diferentes palabras BASIC.

Vamos a comenzar por estudiar la sentencia LET.

El formato general de LET es el siguiente:

LET <nombre de variable> = <valor>

Hay que advertir que el signo = no implica igualdad matemática. Representa que el valor de la derecha es asignado a la variable de la izquierda, de modo que queda almacenado en un espacio de la memoria.

Lógicamente el valor que asignamos a la variable dependerá del tipo de variable utilizada. Si la variable es alfanumérica le podremos asignar una cadena u otra variable alfanumérica previamente asignada. No podemos asignar valores numéricos. Si, por el contrario, la variable es de tipo numérico, podremos asignarle constantes numéricas, expresiones aritméticas o cualquier otra variable numérica que ya tenga un valor previamente asignado.

Veamos algunos ejemplos:

```
10 LET A = 5
```

Esta línea significa que al ejecutar el programa se reserva un espacio en la memoria, cuyo nombre es la variable A y cuyo contenido es un 5.

```
20 LET A = A + 1
```

Esta línea indica que se suma un 1 al contenido de la variable A y el resultado se guarda de nuevo en A, con lo cual se pierde su antiguo valor, ya que en una variable no se puede almacenar más de un dato simultáneamente.

Si queremos imprimir en pantalla el último valor que toma la variable A, tendremos que añadir la línea:

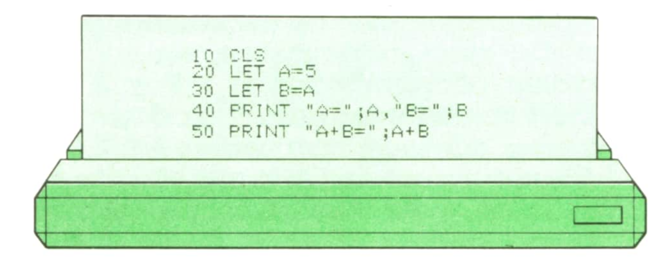
```
30 PRINT A
```

Tenemos aquí un nuevo formato para PRINT:

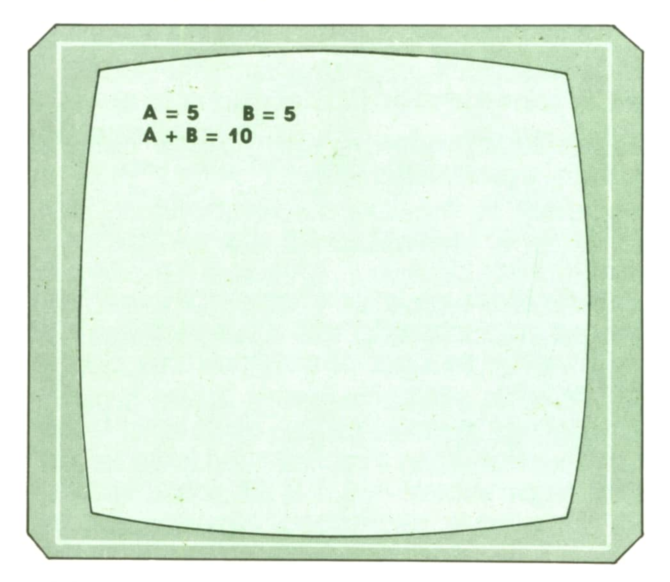
PRINT <nombre de variable>

Los nombres de variables nunca van entre comillas (eso sería una cadena) y lo que hace la instrucción PRINT es imprimir en pantalla el contenido de la variable, es decir, el valor asignado a la variable. Por tanto, con el programa que hemos desarrollado aparecerá en pantalla un 6.

Por otra parte, también podemos asignar a una variable el contenido de otra, tal y como podemos ver en el programa 1.



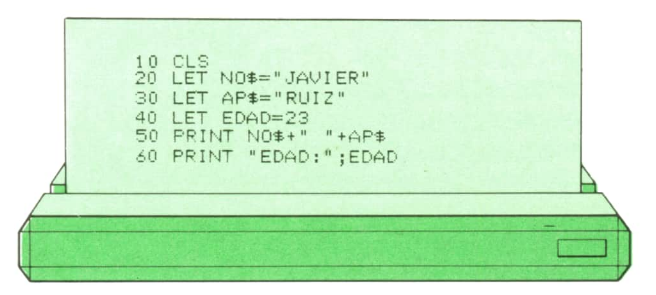
En la línea 30 asignamos a B el contenido de A, es decir, un cinco, aunque A sigue conservando su valor. Después, combinando adecuadamente cadenas y variables, separándolas con coma, o punto y coma según convenga, obtenemos una impresión en pantalla similar a la de la figura 3.





Presentación en pantalla del programa 1.

En la línea 50 podemos observar que, una vez que las variables tienen un valor asignado, podemos realizar operaciones con ellas apareciendo en pantalla el resultado. Análogamente podemos utilizar las variables alfanuméricas, como el programa 2.



Normalmente los ordenadores asignan el valor cero a una variable numérica si previamente no se le ha asignado ningún valor. De igual modo proceden con las variables alfanuméricas a las que les asignan la cadena vacía (""). Si tecleamos:

PRINT HOLA

En pantalla aparecerá un cero. Sin embargo, esto no sucede en el SPECTRUM, que imprimiría un mensaje de error: VARIABLE NOT FOUND, ya que necesita inicializar todas las variables que se vayan a utilizar.

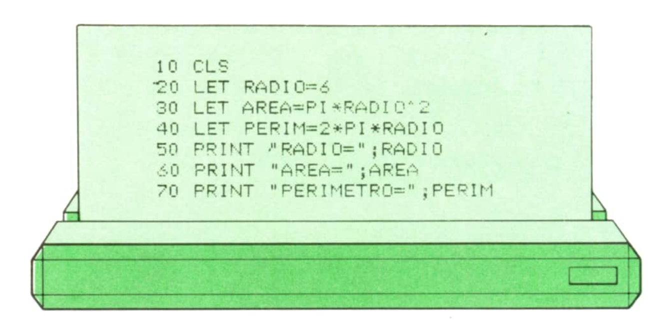
Por otra parte, en todos los ordenadores, excepto en el SPECTRUM, se puede suprimir la palabra LET. Por ejemplo, la instrucción:

10 LADO = 8

es perfectamente válida, ya que tiene el mismo significado que:

10 LET LADO = 8

Finalmente, vamos a desarrollar un programa que permita calcular el área y el perímetro de una circunferencia de radio 6.



En la línea 30 asignamos a la variable AREA el resultado de la operación:

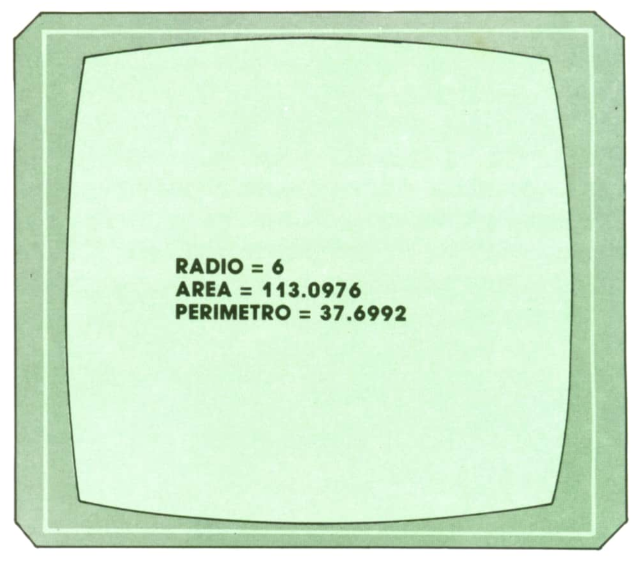
3.1416*6 2

por tanto, PI tiene ya un valor asignado en el sistema (3.14159...). Esto no sucede en todos los ordenadores; por tanto, si nuestro ordenador no tiene asignado el valor de PI, tendremos que añadir al programa la línea:

25 LET PI = 3.1416

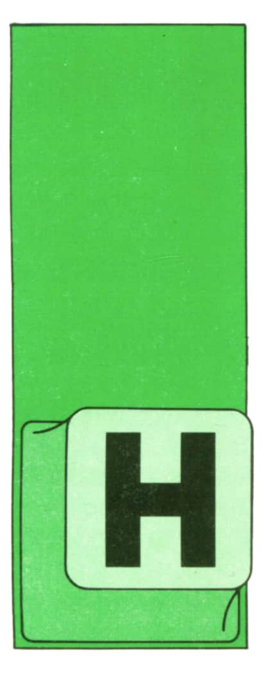
En la línea 40 procedemos análogamente asignando a la variable PERIM la expresión aritmética que permite calcular la longitud de la circunferencia.

Por último, las líneas 50, 60 y 70 se encargan de imprimir los resultados en pantalla, tal y como muestra la figura 4.





Presentación en pantalla del programa 3.



MAQUINA 6502

COMMODORE 64

HEMOS visto hasta ahora cómo funciona el microprocesador del COMMODORE 64, es decir, la parte activa del ordenador que, en última instancia, es la que decide su comportamiento.

Si no lo ha entendido todo, no se preocupe, basta con que se haya quedado con esta idea básica: se trata de una estructura compleja de unidades de memoria en la cual se controla la forma en la que se transfieren los datos, así como la realización de acciones tales como sumas, restas, operaciones lógicas, etc.

Para ello dispone de unos registros propios, que utiliza para llevar a cabo y contabilizar dichas operaciones.

Ahora vamos a dar un paso atrás y a presentar, de una manera simple, lo que llamamos "lenguaje máquina" o "código máquina".

Se intentará responder y dar las nociones generales acerca de una serie de cuestiones que podríamos resumir así:

— ¿Qué es el lenguaje máquina y por qué se utiliza tanto?
— Sistemas de numeración.
— Comandos o nemotécnicos utilizados en la programación en lenguaje máquina.
— Rutinas de interés en lenguaje máquina.

El objetivo final es que el lector se familiarice con las palabras, comandos y rutinas más utilizados en este lenguaje, de tal forma que más adelante, cuando se encuentre ante un libro o revista más avanzados sobre el tema, no lo cierre al ver que no entiende nada, y piense que el lenguaje máquina no está a su alcance.

Siga leyendo y verá cómo con un poco de paciencia y buena voluntad podrá entender y hacer algunas "cosillas" interesantes en lenguaje máquina.



¿Qué es el lenguaje máquina y para qué sirve?

Cuando encendemos nuestro COMMODORE 64 vemos que aparece un mensaje en pantalla que nos indica que está listo para recibir instrucciones. Su COMMODORE 64 ya tiene el lenguaje BASIC, y cuando se le introduce un comando, él dispone de un interpretador del BASIC que está incorporado al sistema operativo, y que se encarga de transformarlo en pasos capaces de ser ejecutados.

Esta labor conlleva una serie de pasos que podrían resumirse así:

— Compararlo con el resto de los comandos BASIC hasta que lo encuentra en su tabla.
— Guardar su posición en dicha tabla.
— Ir a la rutina correspondiente a ese comando.
— Ejecutar el comando.

Si pudiéramos ahorrarnos la labor del intérprete y ejecutar directamente el comando en cuestión, ganaríamos un tiempo precioso, que será mayor cuanto más complejos sean los comandos a ejecutar.

Para esta programación directa es para lo que se utiliza el lenguaje máquina, el cual permite una rapidez que puede ser entre 10 y 1.000 veces superior a su equivalente programa escrito en BASIC.

Pero hay algo más que la *velocidad*, que hace al lenguaje máquina casi imprescindible.

Existen ciertos casos que no se pueden hacer en BASIC, como son aquellas rutinas que utilizan la técnica de INTERRUPTS.

Sin entrar en detalles que no serían de utilidad en esta sección, diremos que un interrupt o interrupción es una función interna del ordenador que ocurre cada sesentaavo de segundo y que se ejecuta "siempre" independientemente de lo que estuviera haciendo en ese momento.

Es decir, cada sesentaavo de segundo su COMMODORE 64 abandona lo que está haciendo, ejecuta la rutina de interrupt y continúa donde se quedó antes de la interrupción. Todo esto pasa inadvertido para el usuario, debido a la gran velocidad con que se ejecuta.

Pues bien, las técnicas de interrupciones se aprovechan de esta característica de su ordenador para introducir sus propias rutinas, de tal forma que las incorporan a la rutina de interrupt. Así cada sesentaavo de segundo su ordenador podrá ejecutar la rutina que haya sido, digamos, "añadida". ¿Interesante, verdad?

Esta es la técnica por la cual a veces en ciertos programas parece que el ordenador hace dos cosas a la vez, como tocar música mientras se está ejecutando el programa. La parte musical ha sido incorporada a las interrupciones y se ejecuta independientemente del resto del programa.

¿Qué hace el ordenador durante una de sus interrupciones?

Hace bastantes cosas, entre las cuales se encuentran la renovación del reloj interno y comprobación del teclado y periféricos.

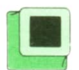
La forma de interceptar las interrupciones la veremos más adelante, pero aquí queda una idea de sus posibilidades.

Otro aspecto del lenguaje máquina es el aprovechamiento del espacio en la memoria, tanto en la creación de un programa como en el almacenamiento de datos. Si el programa está bien estructurado, será bastante más corto y compacto que el correspondiente en BASIC.

Por todo ello, casi siempre merece la pena esforzarse y aprender a programar en lenguaje máquina, porque los resultados compensan el esfuerzo.

Muchas veces lo que se hace es un programa en BASIC, el cual lleva alguna rutina en lenguaje máquina a la que se llama mediante el comando SYS. Así se combinan la claridad externa del BASIC con la rapidez del lenguaje máquina dentro del mismo programa.

Una vez vista y entendida la razón de ser del lenguaje máquina, pasemos a conocer las herramientas que va a necesitar a la hora de hacer sus programas.



Sistemas de numeración

Estamos acostumbrados a manejar el sistema decimal para hacer cualquier tipo de operación con números. Como sabemos, este sistema se compone de 10 dígitos diferentes; 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Pero, desgraciadamente, el ordenador trabaja sólo con "unos" y "ceros", es decir, utiliza el sistema binario. Por ello, es necesario saber operar con este sistema, y transformar un número decimal en binario, y viceversa.

Asimismo, conviene también familiarizarse con el sistema hexadecimal que utiliza 16 dígitos diferentes: 0, 1, 2, 3, 4, 5,

Veamos todo esto con un ejemplo gráfico:

Si tomamos el número decimal 124 y lo descomponemos nos queda:

$124 = 4 \times 10^0 + 2 \times 10^1 + 1 \times 10^2 =$
$= 4 + 20 + 100$

¿A qué número binario correspondería? Muy fácil, hagamos divisiones consecutivas por dos, hasta que el cociente sea menor que dos.

124 : 2 = 62 Resto 0
62 : 2 = 31 Resto 0
31 : 2 = 15 Resto 1
15 : 2 = 7 Resto 1
7 : 2 = 3 Resto 1
3 : 2 = 1 Resto 1

Ahora tomamos el último cociente y todos los restos en orden inverso al que han sido obtenidos y nos queda: 1111100

Para comprobar que éste es el número 124 en decimal, vamos a descomponerlo en potencias de dos:

$1111100 = 0 \times 2^0 + 0 \times 2^1 + 1 \times 2^2 + 1 \times 2^3$
$+ 1 \times 2^4 + 1 \times 2^5 + + 1 \times 2^6 = 0 + 0 + 4 + 8$
$+ 16 + 32 + 64 = 124$

Intentemos ahora la transformación del número 124 decimal al sistema hexadecimal.

Para ello hacemos divisiones sucesivas por 16 hasta obtener un cociente menor que 16.

124 : 16 = 7 Resto 12

Ahora tomamos el último cociente y todos los restos obtenidos en orden inverso.

Si tenemos en cuenta que el número 12 es inmediato en hexadecimal C, queda:

124	=	7C	=	1111100
Decimal		Hexadecimal		Binario

A partir de ahora cualquier número binario lo compondremos de ocho dígitos e irá precedido del símbolo %.

Esto es debido a que cada una de las unidades de memoria o registros de nuestro ordenador se compone de ocho "conexiones" que veremos más tarde, y en cada una de ellas sólo se distinguen dos estados: encendido (1) y apagado (0). Por ello, representaremos siempre ocho dígitos binarios, que pueden ser 1 ó 0.

Así, pues, el número anterior 124 en decimal quedaría como: %01111100 en binario.

Aquí nos surge una pregunta; ¿cuál es el mayor número que se puede representar con ocho dígitos binarios? Sencillo, aquél con todos los dígitos 1.

Veámoslo:

$\%11111111 = 1 \times 2^0 + 1 \times 2^1 + 1 \times 2^2 + 1 \times 2^3 + 1 \times 2^4 + + 1 \times 2^5 + 1 \times 2^6 + 1 \times 2^7 = 1 + 2 + 4 + 8 + 16 + + 32 + 64 + 128 = 255$

Esto nos lleva a la conclusión de que en un registro existen 255 posibilidades o combinaciones diferentes, que van

	0	decimal
desde la	%00000000	binario
	%00	hexadecimal
	255	decimal
hasta la	%11111111	binario
	$FF	hexadecimal

Esto es fácilmente comprobable sin más que preguntándole al ordenador qué número almacena en una determinada posición de memoria X, mediante la función PRINT PEEK(X). Siempre se obtendrá como resultado un número entre 0 y 255.

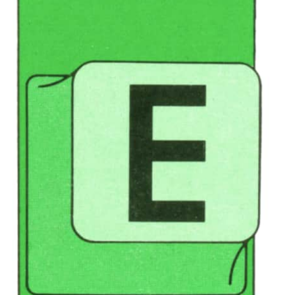
A continuación, y para terminar con los sistemas de numeración, puede ser útil disponer de una tabla de conversión decimal-hexadecimal para valores comprendidos entre 0 y 255.

Dec.	Hex.	Dec.	Hex.	Dec.	Hex.	Dec.	Hex.	Dec.	Hex.
0	00	52	34	103	67	154	9A	205	CD
1	01	53	35	104	68	155	9B	206	CE
2	02	54	36	105	69	156	9C	207	CF
3	03	55	37	106	6A	157	9D	208	D0
4	04	56	38	107	6B	158	9E	209	D1
5	05	57	39	108	6C	159	9F	210	D2
6	06	58	3A	109	6D	160	A0	211	D3
7	07	59	3B	110	6E	161	A1	212	D4
8	08	60	3C	111	6F	162	A2	213	D5
9	09	61	3D	112	70	163	A3	214	D6
10	0A	62	3E	113	71	164	A4	215	D7
11	0B	63	3F	114	72	165	A5	216	D8
12	0C	64	40	115	73	166	A6	217	D9
13	0D	65	41	116	74	167	A7	218	DA
14	0E	66	42	117	75	168	A8	219	DB
15	0F	67	43	118	76	169	A9	220	DC
16	10	68	44	119	77	170	AA	221	DD
17	11	69	45	120	78	171	AB	221	DE
18	12	70	46	121	79	172	AC	223	DF
19	13	71	47	122	7A	173	AD	224	E0
20	14	72	48	123	7B	174	AE	225	E1
21	15	73	49	124	7C	175	AF	226	E2
22	16	74	4A	125	7D	176	B0	227	E3
23	17	75	4B	126	7E	117	B1	228	E4
24	18	76	4C	127	7F	178	B2	229	E5
25	19	77	4D	128	80	179	B3	230	E6
26	1A	78	4E	129	81	180	B4	231	E7
27	1B	79	4F	130	82	181	B5	232	E8
28	1C	80	50	131	83	182	B6	233	E9
29	1D	81	51	132	84	183	B7	234	EA
30	1E	82	52	133	85	184	B8	235	EB
31	1F	83	53	134	86	185	B9	236	EC
32	20	84	54	135	87	186	BA	237	ED
33	21	85	55	136	88	187	BB	238	EE
34	22	86	56	137	89	188	BC	239	EF
35	23	87	57	138	8A	189	BD	240	F0
36	24	88	58	139	8B	190	BE	241	F1
37	25	89	59	140	8C	191	BF	242	F2
38	26	90	5A	141	8D	192	C0	243	F3
39	27	91	5B	142	8E	193	C1	244	F4
40	28	92	5C	143	8F	194	C2	245	F5
41	29	93	5D	144	90	195	C3	246	F6
42	2A	94	5E	145	91	196	C4	247	F7
43	2B	95	5F	146	92	197	C5	248	F8
44	2C	96	60	147	93	198	C6	249	F9
45	2D	97	61	148	94	199	C7	250	FA
46	2E	98	62	149	95	200	C8	251	FB
47	2F	99	63	150	96	201	C9	252	FC
48	30	100	64	151	97	202	CA	253	FD
49	31	101	65	152	98	203	CB	254	FE
50	32	102	66	153	99	204	CC	255	FF
51	33								



PROGRAMAS EDUCATIVOS
PROGRAMAS DE UTILIDAD
PROGRAMAS DE GESTION
PROGRAMAS DE JUEGOS

Programa: Dump de memoria



STE pequeño programa que aparece a continuación nos servirá para testear la memoria y para buscar dentro de ésta cualquier cosa que necesitemos. DUMP significa algo así como VOLCAR. Según esto, lo que hace este programa es volcar el contenido de la memoria en la pantalla.

La rutina nos preguntará la dirección de inicio, la dirección final y, posición a posición de memoria, nos irá mostrando en la pantalla los siguientes datos:

— Dirección de memoria que estamos mirando.

— Contenido de dicha posición en decimal.

— Contenido de dicha posición en hexadecimal.

— El carácter que tiene el código ASCII del valor almacenado en dicha posición.

Sobre los tres primeros datos no hace falta explicar nada, ya que son autoexplicativos. Pero sobre el cuarto hay que decir que, a veces, no necesitamos saber el valor numérico de una posición de memoria, sino el carácter que tiene como código ASCII dicho valor numérico. Este programa está preparado para sacarnos por pantalla el carácter que tiene el código ASCII de cada posición de memoria. En el caso de que dicho código ASCII sea menor de 32, como no se puede imprimir sin estropear el formato de la pantalla, se imprimirá un asterisco (*). En el caso del SPECTRUM tampoco se imprimen los caracteres con código ASCII mayor de 164, ya que a partir de este carácter lo único que aparecería por pantalla serían los TOLKENS del SPECTRUM.

El programa 1 está preparado sólo para los usuarios del SPECTRUM, pero el 2 sirve para todos los demás ordenadores.

```
 1 REM ***********************
 2 REM *   DUMP DE MEMORIA   *
 3 REM ***********************
 4 REM *POR J. GARCIA LUENGO *
 5 REM ***********************
 6 REM *(c)Ed. Siglo Cultural*
 7 REM *(c)1987              *
 8 REM ***********************
 9 REM
10 POKE 23658,8
11 FOR A=USR "A"+7 TO USR "U"+7 STEP 8
12 POKE A,255
13 NEXT A
20 INPUT "DIRECCION INICIO ?";DI
30 FOR A=DI TO 65535 STEP 21
```

```
  40 CLS
  50 PRINT "DIR.    DECIM.  HEXAD.     ASCII";AT 0,0; OVER 1;"___________________
______________"; OVER 0
  60 FOR B=0 TO 20
  70 LET PD=PEEK (A+B)
  80 LET P1=PD/16
  90 LET P=INT P1
 100 LET LO=LEN STR$ (A+B)
 110 PRINT STR$ (A+B)+("........" AND LO=1)+("......." AND LO=2)+("......" AND L
O=3)+("....." AND LO=4)+("...." AND LO=5);
 115 PRINT STR$ (PD)+"........"( TO 8-(PD>9)-(PD>99));
 120 PRINT (STR$ P AND P<10)+("A" AND P=10)+("B" AND P=11)+("C" AND P=12)+("D" A
ND P=13)+("E" AND P=14)+("F" AND P=15);
 130 LET P2=PD-P*16
 140 LET P=P2
 150 PRINT (STR$ P AND P<10)+("A" AND P=10)+("B" AND P=11)+("C" AND P=12)+("D" A
ND P=13)+("E" AND P=14)+("F" AND P=15);
 160 PRINT "..........";(CHR$ PD AND (PD>31 AND PD<164))+("*" AND (PD<32 OR PD>1
64))
 170 NEXT B
 180 PRINT #1; FLASH 1;"P=PARAR C=COPY"
 190 PAUSE 0
 200 LET K$=INKEY$
 210 IF K$="P" THEN  CLEAR : GO TO 20
 220 IF K$="C" THEN  COPY
 230 NEXT A
```

```
 10 REM ******************************
 20 REM *      DUMP DE MEMORIA       *
 30 REM ******************************
 40 REM
 50 REM ******************************
 60 REM * POR Fco. Morales Guerrero *
 70 REM ******************************
 80 REM
 90 REM ******************************
 100 REM * (c) Ed. Siglo Cultural    *
 110 REM * (c) 1987                  *
 120 REM ******************************
 130 REM
 140 DEF FNP$=MID$(A$,1+INT(PP/16),1)+MID$(A$,1+PP-INT(PP/16)*16,1)
 145 LET A$="0123456789ABCDEF"
 150 CLS
 160 LOCATE 10,1
 170 INPUT "Desde la direccion ... ";D1
 180 LOCATE 12,1
 190 INPUT "Hasta ... ";D2
 200 IF D2<D1 OR D1<1 OR D2>65535! THEN GOTO 150
 210 FOR I=D1 TO D2 STEP 20
 220    CLS
 230    PRINT " DIR.   DECIMAL   HEXADECIMAL    ASCII"
 240    PRINT "---------------------------------------"
 250    FOR J=I TO I+19
 260       LET PP=PEEK(J)
```

```
270        LET P$=FNP$
280        IF PP<32 THEN LET C$="*":GOTO 300
290        LET C$=CHR$(PP)
300        PRINT USING "##### ... ###";J;PP;:PRINT " ....... ";P$;" .......... ";
C$
310     NEXT   J
320 PRINT
330 PRINT "(C) COPIAR                        (P) PARAR";
350 LET B$=INKEY$
360 IF B$="" THEN GOTO 350
370 IF B$="C" OR B$="c" THEN GOSUB 1000
380 IF B$="P" OR B$="p" THEN GOTO 500
390 NEXT I
500 CLS
510 END
1000 REM
1010 REM *** SALIDA POR IMPRESORA ***
1020 REM
1030 LPRINT " DIR.   DECIMAL    HEXADECIMAL    ASCII"
1040 LPRINT "---------------------------------------"
1050 FOR J=I TO I+19
1100     LPRINT USING "##### ... ###";J;PP;:PRINT " ....... ";P$;" .......... ";C
$
1110 NEXT   J
1120 RETURN
```

Las variaciones que hay que realizar para que el programa funcione en ordenadores distintos del IBM son las siguientes:

COMMODORE:

```
150 PRINT CHR$(147)
160 POKE 214,10:POKE 211,0
180 POKE 214,12:POKE 211,0
220 PRINT CHR$(147)
330 PRINT J; TAB(10); PP; TAB(22); P$; TAB(35); C$
350 GET B$
500 PRINT CHR$(147)
1025 OPEN 1,4
1027 CMD 1
1030 PRINT "DIR. DECIMAL HEXADECIMAL ASCII"
1040 PRINT "-------------"
1060 PRINT J; TAB(10); PP; TAB (22); P$; TAB (35); C$
1115 CLOSE 1
```

AMSTRAD:

```
160 LOCATE 10,1
180 LOCATE 12,1
```

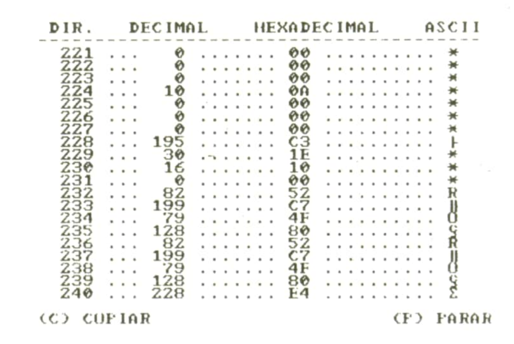
MSX:

```
160 LOCATE 10,1
180 LOCATE 12,1
```

DIR.	DECIM.	HEXAD.	ASCII
50	97	61	a
51	92	5C	\
52	229	E5	*
53	195	C3	*
54	158	9E	*
55	22	16	*
56	245	F5	*
57	229	E5	*
58	42	2A	*
59	120	78	x
60	92	5C	\
61	35	23	#
62	34	22	"
63	120	78	x
64	92	5C	\
65	124	7C	|
66	181	B5	*
67	32	20	
68	3	03	*
69	253	FD	*
70	52	34	4

P=PARAR C=COPY

Ejemplo de ejecución del programa «DUMP de memoria» en el SPECTRUM.



DIR.	DECIMAL	HEXADECIMAL	ASCII
221	0	00	*
222	0	00	*
223	0	00	*
224	10	0A	*
225	0	00	*
226	0	00	*
227	0	00	*
228	195	C3	├
229	30	1E	*
230	16	10	*
231	0	00	*
232	82	52	R
233	199	C7	╟
234	79	4F	O
235	128	80	Ç
236	82	52	R
237	199	C7	╟
238	79	4F	O
239	128	80	Ç
240	228	E4	Σ

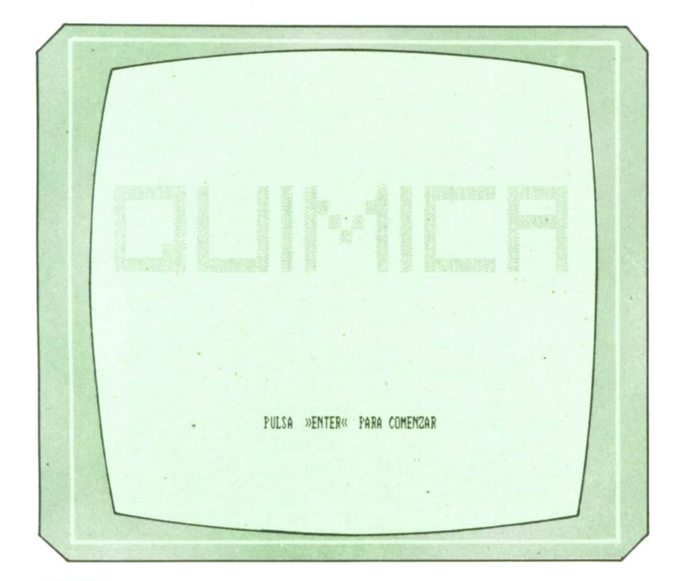
(C) COPIAR (P) PARAR

Ejemplo de ejecución del programa "DUMP de memoria" en el IBM.



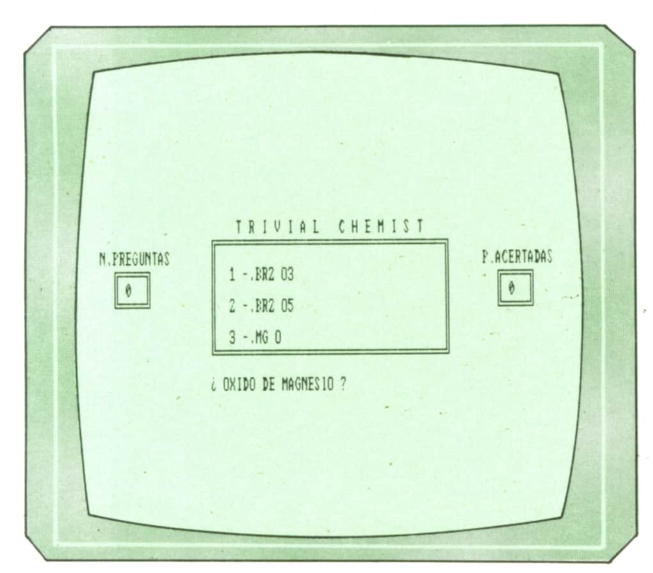
Programa: Trivial química

El programa que proponemos a continuación, aunque es válido sólo para los usuarios de IBM y compatibles, volverá a aparecer en tomos posteriores, de forma que valga para los demás ordenadores.



Pantalla de presentación del programa «Trivial química».

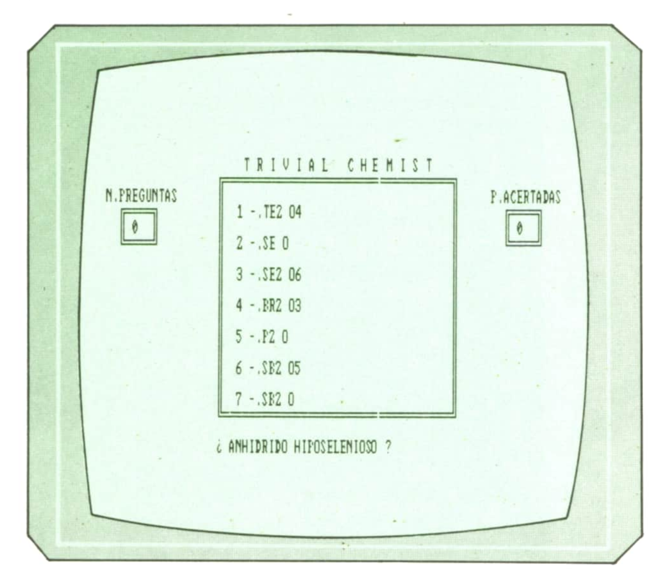
El programa es eminentemente educativo. Nos permitirá repasar y aprender la formulación de los óxidos y de los anhídridos. Para hacer el programa más agradable y menos árido se ha organizado el programa de forma que, para una pregunta, aparecen en la pantalla una serie de respuestas entre las cuales está la verdadera. El número de respuestas que aparecen en pantalla por cada pregunta varía con el grado de dificultad del programa.





Pantalla en un nivel bajo de dificultad.

El usuario del programa puede elegir entre cinco niveles distintos de dificultad. Cada nivel se diferencia del anterior en que en la pantalla aparecen un número más alto de posibles respuestas.





Pantalla en un nivel alto de dificultad.

Dentro de cada nivel de dificultad se puede elegir el número de preguntas entre 1 y 82. El usuario puede elegir el número de preguntas que desea antes de comenzar el examen.

Al final del programa, cuando ha sido agotado el número de respuestas prescrito por el usuario, el programa da un resumen del examen. En dicho resumen se especifican los siguientes datos:

— Número de preguntas que se han hecho.

— Número de respuestas acertadas.

— Tanto por ciento (%) de respuestas correctas.

— Valoración de la prueba de 1 a 10.

Durante la ejecución del programa aparece continuamente en pantalla el número de respuestas que se le han hecho al usuario y cuántas de éstas han sido contestadas correctamente.

En próximos tomos aparecerán nuevos programas de formulación química. En ellos se verá la formulación de hidróxidos, ácidos (tanto hidrácidos como oxácidos), sales, sales ácidas, etc.

T R I V I A L C H E M I S T

RESULTADO DEL CONTROL

NUMERO DE PREGUNTAS......... 10

PREGUNTAS ACERTADAS......... 7

PORCENTAJE QUIMICO.......... 70 %

NOTA DEL CONTROL............ 7

¿ QUIERES HACER OTRO EXAMEN (S/N) ?

El programa nos dice qué tal hemos hecho la prueba.

```
10 REM **********************************************************************
11 REM **************** TRIVIAL CHEMIST POR CARLOS DORAL *******************
12 REM **********************************************************************
13 REM
14 REM
15 REM **********************************************************************
16 REM ************    (C) EDICIONES SIGLO CULTURAL  (1987)    **************
17 REM **********************************************************************
18 REM
19 SCREEN 0
20 COLOR 6
21 CLS
22 DIM A(82)
23 DIM P$(82)
24 DIM R$(82)
25 FOR F=1 TO 82
26    READ P$(F)
27 NEXT F
28 FOR F=1 TO 82
29    READ R$(F)
30 NEXT F
31 LOCATE 4,1
32 PRINT "   ~~~~~~~~~~   ~~        ~~ ~~~~ ~~           ~~ ~~~~  ~~~~~~~~~
~~~~~~~ "
33 PRINT "  ~~        ~~  ~~        ~~  ~~  ~~~~       ~~~~  ~~  ~~        ~~
      ~~"
34 PRINT "  ~~        ~~  ~~        ~~  ~~  ~~  ~~  ~~   ~~  ~~  ~~
      ~~"
35 PRINT "  ~~        ~~  ~~        ~~  ~~  ~~     ~~    ~~  ~~  ~~
      ~~"
36 PRINT "  ~~    ~~  ~~  ~~        ~~  ~~  ~~     ~~    ~~  ~~  ~~
~~~~~~~~~"
37 PRINT "  ~~      ~~~~  ~~        ~~  ~~  ~~           ~~  ~~  ~~        ~~
      ~~"
38 PRINT "   ~~~~~~~~~~    ~~~~~~~~~~  ~~~~ ~~           ~~ ~~~~  ~~~~~~~~~
      ~~"
39 FOR C=1 TO 50
```

```
40   SOUND 100+INT(RND*800),1
41 NEXT C
42 LOCATE 20,26
43 PRINT "PULSA / ENTER . PARA COMENZAR"
44 IF INKEY$<>CHR$(13) THEN GOTO 44
45 LOCATE 20,20
46 LET ER=0
47 RANDOMIZE TIMER
48 INPUT "    (    NIVEL DE DIFICULTAD(1-5)   ";A$
49 GOSUB 171
50 IF ER=1 THEN 45
51 LET ND=VAL(A$)
52 IF ND>5 OR ND<1 THEN GOTO 45
53 LET P=0
54 LET B=0
55 LET Y=10+(ND*2)
56 LOCATE 20,21
57 INPUT "( CUANTAS PREGUNTAS QUIERES (1-82)  ";A$
58 LET ER=0
59 GOSUB 171
60 IF ER=1 THEN GOTO 56
61 LET N=VAL(A$)
62 IF N<1 OR N>82 THEN GOTO 56
63 LOCATE 20,20
64 COLOR 20
65 PRINT "  - ATENCION EMPIEZAN LAS PREGUNTAS !
66 COLOR 6
67 FOR F=1 TO 5
68    FOR X=1 TO 500
69    NEXT X
70    BEEP
71 NEXT F
72 DIM H(ND+3)
73 DIM Q$(ND+3)
74 FOR W=1 TO N
75 CLS
76 LET X1=20
77 LET X2=56
78 LET Y1=3
79 LET Y2=8+(ND*2)
80 GOSUB 363
81 LET X1=5
82 LET X2=10
83 LET Y1=5
84 LET Y2=7
85 GOSUB 363
86 LET X1=64
87 LET X2=69
88 LET Y1=5
89 LET Y2=7
90 GOSUB 363
91 LOCATE 4,3
92 PRINT "N.PREGUNTAS"
93 LOCATE 4,62
94 PRINT "P.ACERTADAS"
95 LOCATE 2,24
96 PRINT "T R I V I A L   C H E M I S T"
97 LOCATE 6,6
98 PRINT P
99 LOCATE 6,65
```

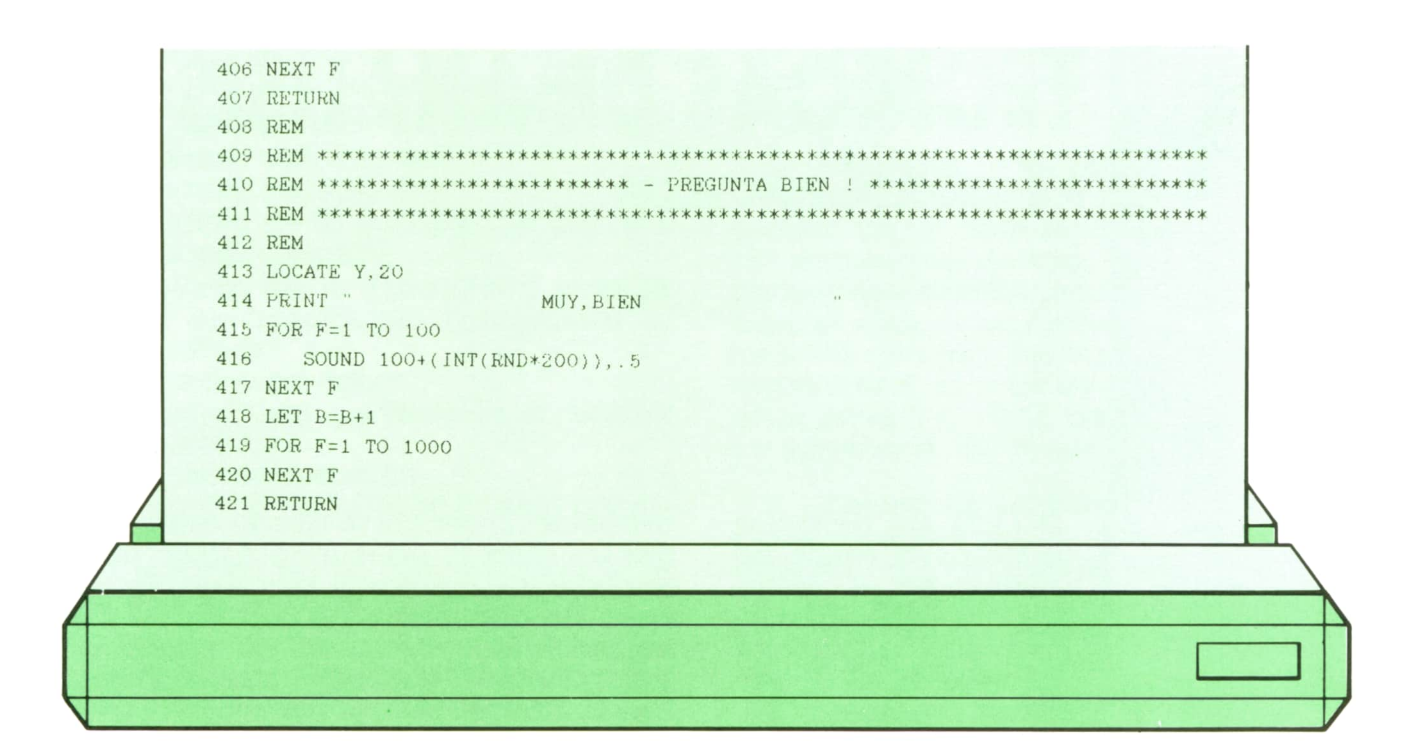
```
100    PRINT B
101    LET C=1
102    LET RN=1+INT(RND*82)
103    FOR F=1 TO 82
104       IF A(F)=RN THEN 102
105    NEXT F
106    LET A(C)=RN
107    FOR F=1 TO ND+3
108       LET H(F)=38+INT(RND*38)
109    NEXT F
110    FOR F=1 TO ND+3
111       FOR C=F+1 TO ND+3
112          IF H(F)=H(C) OR H(F)=RN THEN GOTO 107
113       NEXT C
114    NEXT F
115    FOR F=1 TO ND+3
116       LET S=H(F)
117       LET Q$(F)=R$(S)
118    NEXT F
119    LET R=1+INT(RND*(ND+2))
120    LET Q$(R)=R$(RN)
121    LET C=1
122    FOR F=5 TO 8+(ND*2) STEP 2
123       LOCATE F,22
124       PRINT C;"- ";Q$(C)
125       LET C=C+1
126    NEXT F
127    LOCATE Y,20
128    PRINT "
129    IF RN<38 THEN M$="OXIDO "+P$(RN):ELSE M$="ANHIDRIDO "+P$(RN)
130    LOCATE Y,20
131    PRINT "( ";M$;" ?"
132    LOCATE Y,24+LEN (M$)
133    LET B$=INKEY$
134    IF B$<"1" OR B$>"7" THEN GOTO 133
135    IF B$<>CHR$ (R+48) THEN GOSUB 391
136    IF B$=CHR$(R+48) THEN GOSUB 409
137    LET P=P+1
138    LOCATE Y,20
139    PRINT "             PULSA UNA TECLA
140    IF INKEY$="" THEN GOTO 140
141 NEXT W
142 CLS
143 LET X1=22
144 LET X2=58
145 LET Y1=5
146 LET Y2=16
147 GOSUB 363
148 LOCATE 3,26
149 PRINT "T R I V I A L   C H E M I S T"
150 LOCATE 6,30
151 PRINT "RESULTADO DEL CONTROL"
152 LOCATE 8,24
153 PRINT "NUMERO DE PREGUNTAS.........";N
154 LOCATE 10,24
155 PRINT "PREGUNTAS ACERTADAS.........";B
156 LOCATE 12,24
157 PRINT "PORCENTAJE QUIMICO..........";INT((B*100)/N);"%"
158 LOCATE 14,24
159 PRINT "NOTA DEL CONTROL............"
160 LOCATE 14,52
```

```
161 COLOR 20
162 PRINT INT((B*10)/N)
163 COLOR 5
164 LOCATE 20,20
165 PRINT "(  QUIERES HACER OTRO EXAMEN (S/N) ?"
166 IF INKEY$="S" OR INKEY$="s" THEN RUN
167 IF INKEY$="N" OR INKEY$="n" THEN END
168 GOTO 166
169 END
170 REM
171 REM *********************************************************************
172 REM ******************* CONTROL DE ERRORES DEL INPUT *********************
173 REM *********************************************************************
174 REM
175 IF LEN(A$)<1 OR LEN(A$)>2 THEN ER=1:RETURN
176 IF ASC(A$)<49 OR ASC(A$)>56 THEN ER=1
177 RETURN
178 REM
179 REM *********************************************************************
180 REM **************** DATAS DE LAS PREGUNTAS DE LOS OXIDOS ****************
181 REM *********************************************************************
182 REM
183 DATA DE LITIO
184 DATA SODICO
185 DATA POTASICO
186 DATA RUBIDICO
187 DATA DE CESIO
188 DATA DE PLATA
189 DATA CUPROSO
190 DATA CUPRICO
191 DATA MERCURIOSO
192 DATA MERCURICO
193 DATA AUROSO
194 DATA AURICO
195 DATA DE MAGNESIO
196 DATA DE BERILIO
197 DATA CALCICO
198 DATA ESTRONCICO
199 DATA BARICO
200 DATA DE RADIO
201 DATA ZINCICO
202 DATA CADMICO
203 DATA FERROSO
204 DATA FERRICO
205 DATA COBALTOSO
206 DATA COBALTICO
207 DATA NIQUELOSO
208 DATA NIQUELICO
209 DATA CROMOSO
210 DATA CROMICO
211 DATA MANGANOSO
212 DATA MANGANICO
213 DATA ALUMINICO
214 DATA DE BORO
215 DATA PLUMBOSO
216 DATA PLUMBICO
217 DATA ESTANNOSO
218 DATA ESTANNICO
219 DATA DE PLATINO
220 DATA DE IRIDIO
221 REM
```

```
222 REM ***************************************************************************
223 REM ************** DATAS DE LAS PREGUNTAS DE LOS ANHIDRIDOS ***************
224 REM ***************************************************************************
225 REM
226 DATA HIPOFLUOROSO
227 DATA FLUOROSO
228 DATA FLUORICO
229 DATA PERFLUORICO
230 DATA HIPOCLOROSO
231 DATA CLOROSO
232 DATA CLORICO
233 DATA PERCLORICO
234 DATA HIPOBROMOSO
235 DATA BROMOSO
236 DATA BROMICO
237 DATA PERBROMICO
238 DATA HIPOYODOSO
239 DATA YODOSO
240 DATA YODICO
241 DATA PERYODICO
242 DATA HIPOSULFUROSO
243 DATA SULFUROSO
244 DATA SULFURICO
245 DATA HIPOSELENIOSO
246 DATA SELENIOSO
247 DATA SELENICO
248 DATA HIPOTELUROSO
249 DATA TELUROSO
250 DATA TELURICO
251 DATA HIPONITROSO
252 DATA NITROSO
253 DATA NITRICO
254 DATA HIPOFOSFOROSO
255 DATA FOSFOROSO
256 DATA FOSFORICO
257 DATA HIPOARSENIOSO
258 DATA ARSENIOSO
259 DATA ARSENICO
260 DATA HIPOANTIMONIOSO
261 DATA ANTIMONIOSO
262 DATA ANTIMONICO
263 DATA HIPOBISMUTOSO
264 DATA BISMUTOSO
265 DATA BISMUTICO
266 DATA CARBONOSO
267 DATA CARBONICO
268 DATA SILICIOSO
269 DATA SILICICO
270 REM
271 REM ***************************************************************************
272 REM *************** DATAS DE LAS RESPUESTAS DE LOS OXIDOS ***************
273 REM ***************************************************************************
274 REM
275 DATA Li2O
276 DATA Na2O
277 DATA K2O
278 DATA Rb2O
279 DATA Cs2O
280 DATA Ag2O
281 DATA Cu2O
282 DATA CuO
```

```
283 DATA Hg2O
284 DATA HgO
285 DATA Au2O
286 DATA Au2O3
287 DATA MgO
288 DATA BeO
289 DATA CaO
290 DATA SrO
291 DATA BaO
292 DATA RaO
293 DATA ZnO
294 DATA CdO
295 DATA FeO
296 DATA Fe2 O3
297 DATA CoO
298 DATA Co2O3
299 DATA NiO
300 DATA Ni2O3
301 DATA CrO
302 DATA Cr2O3
303 DATA MnO
304 DATA Mn2O3
305 DATA Al2O3
306 DATA B2O3
307 DATA PbO
308 DATA PbO2
309 DATA SnO
310 DATA SnO2
311 DATA PtO2
312 DATA IrO2
313 REM
314 REM ***********************************************************************
315 REM ************** DATAS DE LAS RESPUESTAS DE LOS ANHIDRIDOS **************
316 REM ***********************************************************************
317 REM
318 DATA F12O
319 DATA F12O3
320 DATA F12O5
321 DATA F12O7
322 DATA Cl2O
323 DATA Cl2O3
324 DATA Cl2O5
325 DATA Cl2O7
326 DATA Br2O
327 DATA Br2O3
328 DATA Br2O5
329 DATA Br2O7
330 DATA I2O
331 DATA I2O3
332 DATA I2O5
333 DATA I2O7
334 DATA SO
335 DATA SO2
336 DATA SO3
337 DATA SeO
338 DATA SeO2
339 DATA SeO3
340 DATA TeO
341 DATA TeO2
342 DATA TeO3
343 DATA N2O
```

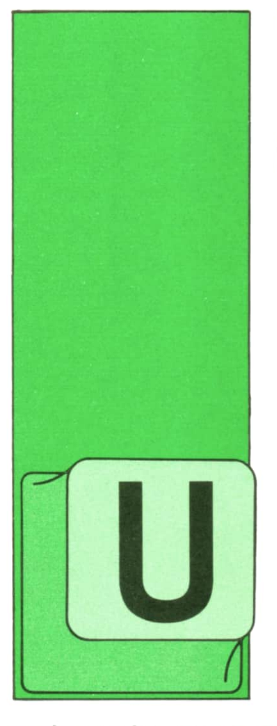
```
344 DATA N2O3
345 DATA N2O5
346 DATA P2O
347 DATA P2O3
348 DATA P2O5
349 DATA As2O
350 DATA As2O3
351 DATA As2O5
352 DATA Sb2O
353 DATA Sb2O3
354 DATA Sb2O5
355 DATA Bi2O
356 DATA Bi2O3
357 DATA Bi2O5
358 DATA Co
359 DATA Co2
360 DATA SiO
361 DATA SiO2
362 REM
363 REM ********************************************************************
364 REM ********************* DIBUJO DE LAS VENTANAS ************************
365 REM ********************************************************************
366 REM
367 LOCATE Y1,X1
368 PRINT CHR$(201)
369 LOCATE Y1,X2
370 PRINT CHR$(187)
371 LOCATE Y2,X1
372 PRINT CHR$(200)
373 LOCATE Y2,X2
374 PRINT CHR$(188)
375 LET F=X1+1
376   LOCATE Y1,F
377   PRINT CHR$(205)
378   LOCATE Y2,F
379   PRINT CHR$(205)
380   LET F=F+1
381   IF F<X2 THEN GOTO 376
382   LET F=Y1+1
383   LOCATE F,X1
384   PRINT CHR$(186)
385   LOCATE F,X2
386   PRINT CHR$(186)
387   LET F=F+1
388   IF F<Y2 THEN GOTO 383
389 RETURN
390 REM
391 REM ********************************************************************
392 REM ************************ - PREGUNTA MALA ! *************************
393 REM ********************************************************************
394 REM
395 LOCATE Y,20
396 PRINT "              MUUYYY,MAAAAL!
397 FOR F=900 TO 100 STEP -30
398   SOUND F,1
399 NEXT F
400 LOCATE Y,20
401 PRINT "  LA RESPUESTA CORRECTA ERA ";R$(RN)
402 COLOR 20
403 LOCATE 3+(R*2),27:PRINT R$(RN)
404 COLOR 5
405 FOR F=1 TO 3000
```



```
406 NEXT F
407 RETURN
408 REM
409 REM ************************************************************************
410 REM ************************* - PREGUNTA BIEN ! ****************************
411 REM ************************************************************************
412 REM
413 LOCATE Y,20
414 PRINT "               MUY,BIEN                "
415 FOR F=1 TO 100
416    SOUND 100+(INT(RND*200)),.5
417 NEXT F
418 LET B=B+1
419 FOR F=1 TO 1000
420 NEXT F
421 RETURN
```

TECNICAS DE ANALISIS

PREVISION DE CONTROLES



UNO de los aspectos básicos a tener en cuenta en el diseño de una aplicación informática cualquiera (incluso en el diseño de un programa individual a realizar en un ordenador personal) es el de los controles y seguridades del proceso. Evidentemente, para que los procesos se realicen correctamente y los resultados sean los esperados, los datos que se suministran al ordenador han de ser correctos y las manipulaciones de los operadores las debidas, pero ha de ser en el proceso de análisis de la aplicación cuando se diseñen unos controles adecuados que aseguren que estos aspectos descritos se realizan debidamente.

Se suelen establecer dos grupos de controles: un primer grupo formado por los que se llaman habitualmente controles físicos y otro formado por los llamados controles lógicos o de contenido.

Con los controles físicos se trata de asegurar la adecuada manipulación física de los documentos y archivos en que se contienen los datos: no debe «perderse» ningún documento ni archivo (cinta magnética, disco, disquete, etc.).

Mediante cotroles lógicos (o de contenido) se confirma la validez de los datos y la ausencia de errores.

A) Entre los controles físicos usualmente establecidos pueden citarse:

1. Presencia de documentos. Normalmente, los datos originales del proceso suelen tomarse de algunos documentos en donde están escritos. Es importante asegurar que el conjunto de documentos que se procesan (que se corresponderá con el conjunto de datos introducidos) es el debido. El control se puede establecer por un número adecuado escrito en el documento, mediante recuento del número de documentos a procesar u otro equivalente.

2. Control físico de archivos. Etiquetas. Es básico que los archivos transportables (soportados en disquetes, discos removibles, etc.) estén controlados en dos sentidos: a) que en cada momento se esté procesando el archivo previsto; b) que se trabaje con la versión adecuada de cada archivo. En efecto, de un mismo fichero existirán normalmente varias versiones (obtenidas en procesos sucesivos) y es importante trabajar con la versión prevista. Esto es especialmente importante cuando los procesos se realizan con intervalos de tiempo pequeños (quizá cada día).

En los procesos de actualización de los ficheros, es usual disponer de tres soportes físicos que se van rotando (se suele llamar a esta organización de «abuelo-padre-hijo»: en cada momento se procesa el «hijo» como entrada y el fichero resultante (en la salida) se escribe sobre el «abuelo»; de este modo si se destruye, por error, alguna información sigue quedando otra tercera copia del fichero («padre») que, aunque no sea la última versión, es bastante válida.

La identificación de los ficheros se realiza mediante etiquetas externas (adhesivas) y mediante etiquetas internas grabadas en el propio soporte físico y que son leídas por el programa en el momento de «abrir» el archivo correspondiente, para comenzar su proceso. Las etiquetas adhesivas se obtienen, a veces, del propio programa (en alguna impresora auxiliar): esta es una buena solución, por la seguridad que aporta. La etiqueta exter-

na debe contener suficiente información para la identificación unívoca del fichero; se suelen incluir los siguientes datos:

— nombre del volumen.
— número del volumen.
— número de la cadena (de proceso a la que pertenece).
— número de la subcadena y de la Unidad de Tratamiento (programa) al que se refiere.
— nombre del fichero.
— número de generación o versión del fichero.
— fecha de creación.
— fecha de caducidad (si está prevista alguna).
— volumen en el que continúa el fichero (si es un fichero multivolumen).

El contenido de las etiquetas internas de los archivos suele venir definido por el sistema en el que se han grabado. Si se puede elegir, conviene incluir aproximadamente la misma información indicada anteriormente y añadir información de los formatos internos del fichero.

B) Los controles lógicos o de contenido más usualmente utilizados son los siguientes:

1. Control de presencia de los datos.
Se debe definir como obligatorio un dato (o un tipo de registro) que siempre deba aparecer para que en el programa se tenga en cuenta esta circunstancia y se controle.

2. Naturaleza de los datos.
Se puede (y debe) controlar el tipo de datos que se introduce en cada registro (y dentro de cada registro en cada campo) para que no se intenten procesar datos numéricos como alfabéticos y viceversa, etc.

3. Control de verosimilitud.
Es decir, que el contenido de cada campo sea el adecuado (y no sólo del *tipo* adecuado).

Es muy normal poder definir el rango de los datos (por ejemplo, si un número puede ser negativo o no, los límites entre los que debe encontrarse, etc.): si es así, debe incluirse este control en el (o los) programa(s).

4. Interrelación de datos.
Es muy normal que la presencia (o ausencia) de un dato signifique la presencia o ausencia de otro. A veces el control hay que establecerlo entre la presencia o el valor de un dato y el rango de valores de otro (por ejemplo, «si el tipo de registro es A, el valor de un campo debe estar entre 1 y 1000; pero si es B, debe ser superior a 1000», etc.).

5. Control de secuencia o de cadencia en la aparición de los datos o de los registros («los números de orden deben ir de cinco en cinco» o «los números deben ser consecutivos y empezar desde 1 al cambiar la fecha, o de año", etc.).

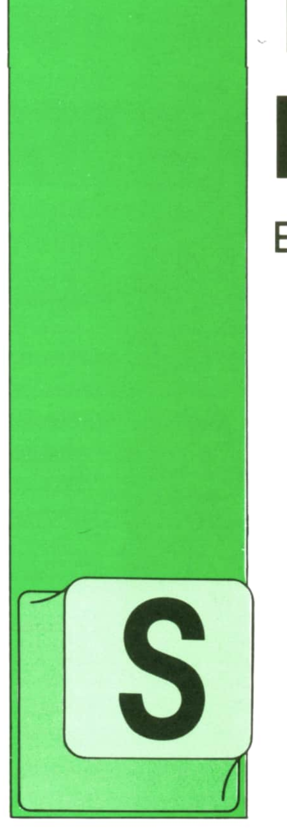
6. Controles calculados.
Es muy común que el propio proceso a realizar con los datos proporcione un control adicional sobre la consistencia de los datos. Por ejemplo, es usual que se introduzcan en un proceso un conjunto de cifras parciales y también su total: en este caso, la simple suma de las cifras individuales y su comparación con el total facilita un control para detectar posibles errores o la ausencia de un dato.

Conviene señalar por último que, aunque habitualmente la naturaleza de los procesos a realizar con los datos o la propia estructura de los datos sugiere el tipo de controles a establecer, si no es así, de todos modos, en el análisis y diseño de las aplicaciones o programas debe preverse la existencia de algún control, de tal modo que se introduzca alguna comprobación (incluso artificialmente) si no aparece ninguna en la naturaleza de las informaciones que se traten.

Además, la situación más común es que en un mismo programa haya que introducir varios de los tipos de control antes indicados e, incluso a veces, interrelacionarlos.

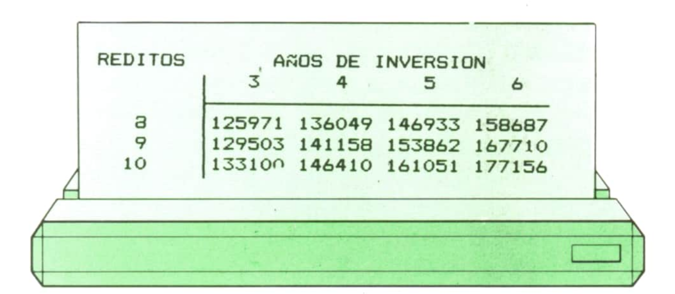
TECNICAS DE PROGRAMACION

ESTRUCTURA DE DATOS



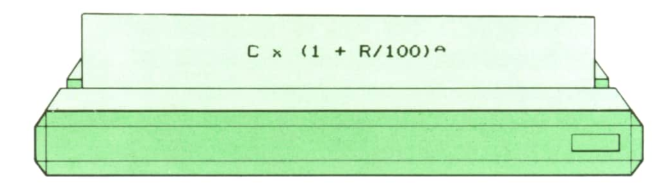
Cálculo de una tabla de interés compuesto

SUPONGAMOS que deseamos invertir una cantidad determinada (por ejemplo, 100.000 pesetas), durante varios años, a cierta tasa de interés compuesto, y deseamos estudiar diversas alternativas. Por ejemplo, podemos realizar la inversión a tres, cuatro, cinco o seis años, y tenemos la posibilidad de elegir entre diversos tipos de interés, tales como el 8, el 9 o el 10 por 100. Podría suceder que fueran posibles todas las combinaciones de los distintos réditos con los años de inversión. Entonces, para poder comparar las diversas alternativas y hacernos una idea de la cantidad que podemos obtener al final del tiempo correspondiente, sería bueno disponer de una tabla que nos presentara dichas cantidades de una forma sencilla de comprender y que salte a la vista. Una tabla como la siguiente:



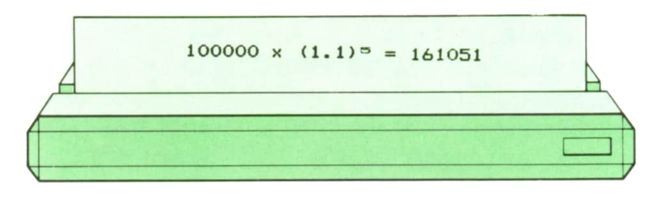
REDITOS	AÑOS DE INVERSION			
	3	4	5	6
8	125971	136049	146933	158687
9	129503	141158	153862	167710
10	133100	146410	161051	177156

Como sabemos, la fórmula que calcula en qué se convierte un capital «C», invertido durante «A» años con una tasa (o rédito) de interés compuesto igual a «R» (expresada en tanto por ciento) es la siguiente:



donde el signo «x» indica multiplicación, el símbolo «/» representa la división y el superíndice formado por la letra A indica que el valor expresado entre paréntesis debe ser elevado a la potencia «A» (es decir, multiplicado «A» veces por sí mismo).

Si en la fórmula anterior sustituimos C por el valor del capital que nos interesa en nuestro caso (100.000 pesetas), R por una de las tasas de interés consideradas (10 por 100) y A por cierto número de años (cinco, por ejemplo), podremos calcular en qué se convierten 100.000 pesetas al 10 por 100 de interés compuesto durante cinco años:



donde hemos redondeado el número obtenido a la peseta más próxima, despreciando los céntimos. Esto significa, por tanto, que nuestras 100.000 pesetas se habrán convertido al cabo de cinco años en 161.051 pesetas o, dicho de otro modo, que hemos obtenido un interés total acumulado de 61.051 pesetas.

Podemos observar que el número obtenido coincide con el que aparece en la

tercera fila y la tercera columna de la tabla dada más arriba, que corresponde, naturalmente, a un rédito del 10 por 100 (cabecera de la fila) y un plazo de cinco años (cabecera de la columna). En efecto, para construir la tabla entera no hay más que repetir la operación anterior para las doce combinaciones posibles de los tres réditos deseados (8, 9 y 10 por 100) y los cuatro plazos posibles (3, 4, 5 y 6 años). Podemos proceder a realizarlos uno tras otro y, de este modo, obtendremos la tabla con cierta facilidad.

Sin embargo, las operaciones que hay que realizar son siempre las mismas. Hacerlas a mano llevaría cierto tiempo, incluso aunque se dispusiera de una calculadora de bolsillo. ¿Por qué no utilizamos el ordenador, que está especialmente preparado para realizar rápidamente y con poco esfuerzo este tipo de operaciones repetitivas? En el capítulo anterior hemos visto que es muy fácil construir tablas en diversos lenguajes de programación. Vamos a aplicar ahora dichas consideraciones a la construcción de la tabla de interés compuesto.

Veamos primero qué clase de programa deseamos construir. Lo haremos lo más general posible, de manera que nos valga cualesquiera que sean los réditos y los plazos que vayamos a colocar como encabezados de las filas y columnas de nuestra tabla. En tal caso, el programa pasará por las siguientes etapas:

1. Leer del teclado el capital.

2. Leer del teclado los réditos deseados, expresados en tantos por ciento.

3. Leer del teclado los plazos deseados, expresados en años.

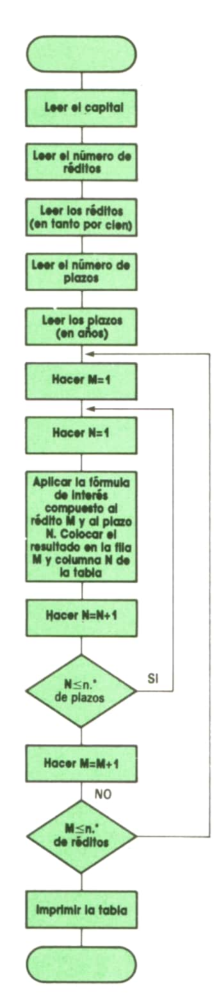
4. Calcular sucesivamente los resultados de aplicar la fórmula de interés compuesto con todas las combinaciones posibles de los réditos y los plazos dados.

5. Rellenar la tabla con los resultados obtenidos.

6. Obtener la tabla por la pantalla.

La figura 1 presenta el diagrama.

Ahora vamos a construir los programas correspondientes en diversos lenguajes de programación. Veamos primero la versión BASIC:




```
10 REM Cálculo de una tabla de interés
   compuesto
20 DIM R(10)
30 DIM A(10)
40 DIM I(10,10)
50 REM Leemos el capital
60 PRINT "Deme el capital"
70 INPUT C
80 REM Leemos los réditos deseados
90 PRINT "Deme el número de réditos"
100 INPUT NR
110 PRINT "Deme los réditos en tanto
    por ciento"
120 FOR N=1 TO NR: INPUT R(N): NEXT N
130 REM Leemos los años deseados
140 PRINT "Deme el número de años"
150 INPUT NA
160 PRINT "Deme los años"
170 FOR N=1 TO NA: INPUT A(N): NEXT N
180 REM Empezamos a calcular la tabla
190 FOR M=1 TO NR
200 FOR N=1 TO NA
210 LET I(M,N)=C*(1+R(M)/100)^A(N)
220 NEXT N: NEXT M
230 REM Ahora escribimos el resultado
240 FOR M=1 TO NR
250 FOR N=1 TO NA
260 PRINT INT(I(M,N));: NEXT N: PRINT:
    NEXT M
```

Las instrucciones 10, 50, 80, 130, 180 y 230, que comienzan con la palabra reservada REM, no son más que comentarios

que el ordenador ignora, pero que a nosotros nos sirven para enterarnos de lo que va haciendo el programa, especialmente si hace mucho tiempo que lo construimos y deseamos volver a utilizarlo o modificarlo para realizar nuevas aplicaciones. Por esta razón es conveniente poner siempre numerosos comentarios en nuestros programas.

Las instrucciones 20 y 30 definen dos "vectores" o series de datos que vamos a utilizar para guardar los réditos (variable R) y los plazos de inversión en años (variable A). Estas variables se declaran mediante la palabra reservada DIM, como vimos en el capítulo tercero. En principio, suponemos que no vamos a desear construir tablas para más de diez réditos distintos o para más de diez plazos diferentes. Por tanto, declaramos que estas series de datos van a tener, precisamente, diez elementos.

La instrucción 40 define la tabla de interés compuesto, como vimos en el capítulo cuarto, también mediante la instrucción DIM, pero dando en este caso el número de filas y el de columnas. La llamamos I, y afirmamos que tendrá diez filas y diez columnas, lo que corresponde con los límites que hemos establecido antes para los réditos y los plazos de inversión.

Las instrucciones 50 a 170 leen del teclado los diversos datos que necesitamos para comenzar a calcular la tabla: el valor del capital (que se guarda en la variable C), los réditos (que pasan a llenar los primeros valores de la variable R), y los plazos de inversión en años (que ocupan los primeros valores de la variable A). Al llegar a la instrucción 180 tendremos, por tanto, todos los datos que necesitamos y podemos comenzar a realizar los cálculos.

Como la tabla tiene dos dimensiones, es preciso utilizar dos bucles (los indicados en la figura 1) para llenar todos sus valores: un bucle para las filas (los réditos) y otro para las columnas (los plazos). En BASIC, construiremos los bucles con instrucciones FOR. Habrá, por tanto, dos de ellas:

```
190 FOR M=1 TO NR
200 FOR N=1 TO NA
210 LET I(M,N)=C*(1+R(M)/100)^A(N)
220 NEXT N: NEXT M
```

La instrucción clave aquí es la número 210. Esta es la que calcula el valor que debe colocarse en un lugar determinado de la tabla. Se observará que no hace otra cosa que aplicar la fórmula del interés compuesto, utilizando los símbolos de BASIC para la multiplicación (*), la división (/) y la elevación de una potencia (^).

Finalmente, las instrucciones 230 a 260 sirven para que aparezcan los valores de la tabla en la pantalla, tal y como se explicó en el capítulo cuarto. Obsérvese que, para despreciar los céntimos, imprimimos INT (I (M, N)), es decir, la parte entera de los valores de la tabla. Esta función no redondea, sino que trunca los valores al entero inmediato inferior.

Al ejecutar el programa anterior obtenemos el siguiente resultado:

```
Deme el capital
? 100000
Deme el número de réditos
? 3
Deme los réditos en tanto por ciento
? 8
? 9
? 10
Deme el número de años
? 4
Deme los años
? 3
? 4
? 5
? 6
 125971  136048  146932  158687
 129502  141158  153862  167710
 133100  146410  161051  177156
```

Veamos ahora la versión PASCAL. El organigrama que vamos a realizar es el mismo que en el caso de BASIC. El programa será el siguiente:

```
program INTERES;
(* Declaración de las variables *)
var
  nr, na, m, n: integer; (* contadores *)
  c:real; (* capital *)
  r, a: array[1..10] of integer; (* réditos y años *)
```

```
    i: array[1..10,1..10] of real; (* tabla de intereses *)
  (* Declaración de la función POTENCIA *)
  function potencia (x:real;n:integer):real;
    (* Devuelve x elevado a n *)
    var i:integer; r:real;
    begin
        r:=1;
        for i:=1 to n do r:=r*x;
        potencia:=r;
    end;
  (* Programa *)
  begin
  (* Pedimos el capital *)
    write('Deme el valor del capital ');
    readln(c);
  (* Pedimos los réditos *)
    write('Deme el número de réditos ');
    readln(nr);
    writeln('Deme los réditos en tanto por ciento ');
    for n:=1 to nr do readln(r[n]);
  (* Pedimos los años *)
    write('Deme el número de años ');
    readln(na);
    writeln('Deme los años ');
    for n:=1 to na do readln(a[n]);
  (* Ya podemos calcular la tabla *)
    for m:=1 to nr do
        for n:=1 to na do
            i[m,n]:=c*potencia((1+r[m]/100),a[n]);
  (* Ahora escribimos los resultados *)
    for m:=1 to nr do
        begin
            for n:=1 to na do write(i[m,n]:6:0,' ');
            writeln;
        end
  end.
```

En este programa también podemos ver numerosos comentarios, que son los textos comprendidos entre los símbolos (* y los símbolos *). Al revés que en BASIC, los comentarios de PASCAL pueden colocarse en cualquier sitio dentro del programa, al principio, en medio o al final de una línea o constituyendo líneas completas.

Hay una diferencia fundamental entre este programa PASCAL y el programa BASIC que hemos visto anteriormente: en PASCAL hemos definido una función especial para elevar un número a una potencia. Esta función es la siguiente:

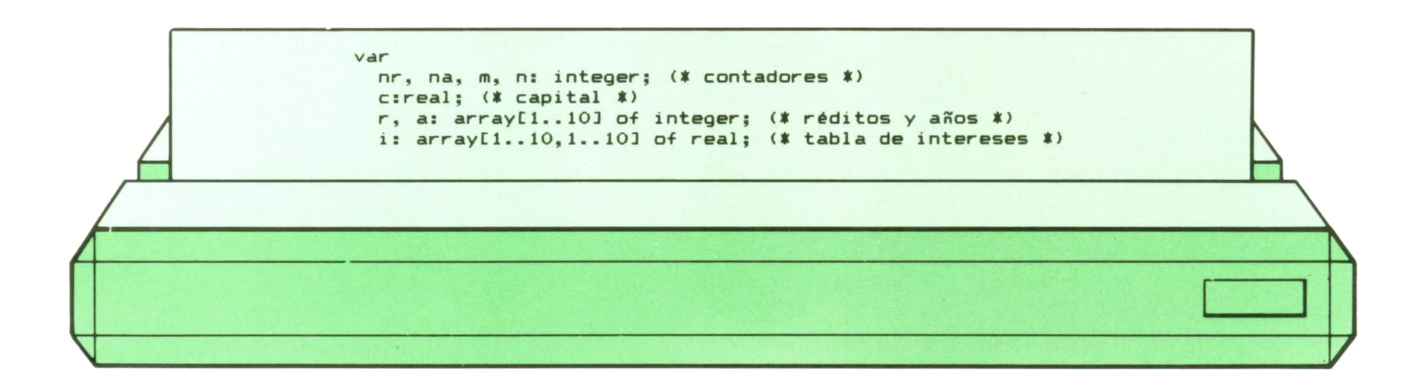
```
function potencia (x:real;n:integer):
  real:
  (* Devuelve x elevado a n *)
  var i:integer; r:real;
  begin
      r:=1;
      for i:=1 to n do r:=r*x;
```



```
      potencia:=r;
  end;
```

Esta función es un "subprograma", es decir, una parte de un programa que se aísla del mismo y realiza una operación determinada (en este caso, multiplica "n" veces por sí mismo el valor de la variable "x"). En capítulos posteriores veremos con mayor detalle esta forma de programar.

Como hemos dicho más de una vez, en PASCAL es preciso declarar todas las variables. La parte declarativa comienza por la palabra reservada VAR, y comprende, además de la función "potencia", las siguientes instrucciones:



El resto del programa es muy semejante a la versión BASIC. También aquí comenzamos por obtener del teclado los valores del capital, los réditos y los plazos en años y efectuamos los cálculos mediante un doble bucle que, en su versión PASCAL, dice así:

```
for m:=1 to nr do
     for n:=1 to na do
          i[m,n]:=c*potencia((1+r[m]/100),
          a[n]);
```

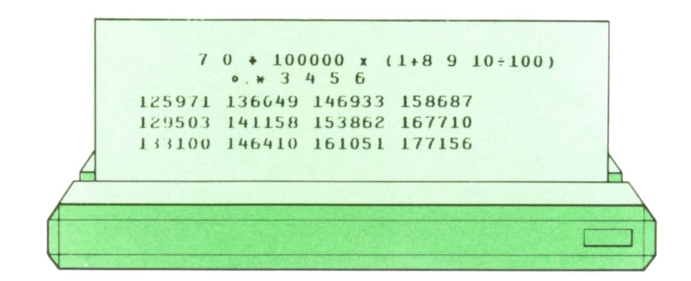
donde puede observarse también la aplicación de la fórmula del interés compuesto, que en este caso utiliza la función "potencia" en lugar de un símbolo predefinido. La última parte del programa PASCAL exhibe el resultado por la pantalla, tal y como se explicó en el capítulo cuarto.

Veamos el resultado de la ejecución de este programa:

```
Deme el valor del capital 100000
Deme el número de réditos 3
Deme los réditos en tanto por ciento
8
9
10
Deme el número de años 4
Deme los años
3
4
5
6
125971 136049 146933 158687
129503 141158 153862 167710
133100 146410 161051 177156
```

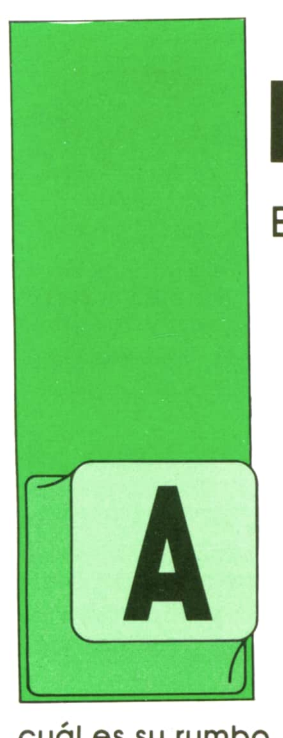
Se observará que, en este caso, se han obtenido valores redondeados hacia la peseta más próxima, por lo que existen diferencias de una peseta en algunos casos con respecto a la solución del programa BASIC.

En APL, dada la ausencia de declaraciones en este lenguaje y su gran facilidad para trabajar con tablas directamente, sin necesidad de bucles, no es necesario utilizar el organigrama anterior, y el programa se reduce a una sola línea. Para interpretarlo, recuérdese lo explicado en el capítulo cuarto sobre la forma de generar tablas en APL, aplicando una operación a dos series de datos en todas las combinaciones posibles.



Se observará que los resultados del programa APL son indénticos a los del programa PASCAL, pues los datos están también redondeados a la peseta más próxima.

En el capítulo primero puede encontrar el lector una versión más completa de este programa APL, que pide del teclado los datos necesarios (capital, tasas de interés y plazos de inversión) y produce una tabla con encabezados de filas y columnas.

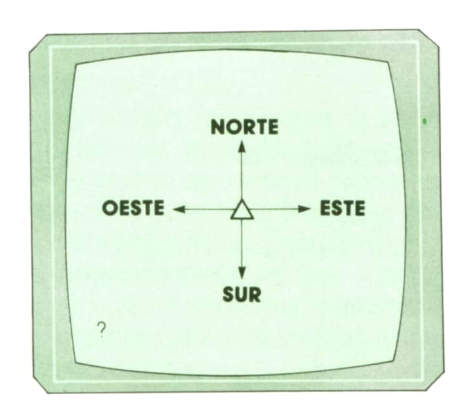


LOGO

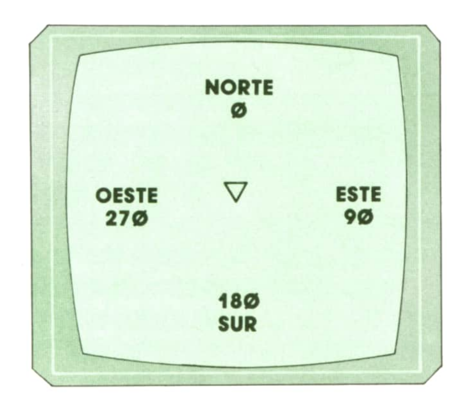
EL RUMBO DE LA TORTUGA

AL igual que cuando viajamos en barco o en avión el piloto sabe perfectamente hacia dónde se dirige, la tortuga cuando se mueve por la pantalla o gira conoce cuál es su rumbo, es decir, en qué dirección y sentido va a realizar su siguiente movimiento.

Para poder hacer esto la tortuga considera que estando en cualquier posición, puede distinguir en la pantalla los cuatro puntos cardinales que todos conocemos:



Cada uno de estos puntos cardinales tiene asignado un valor, así:



Pues bien, podemos utilizar el comando

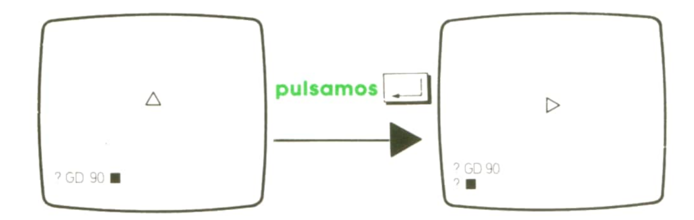
PONRUMBO n

para decirle a la tortuga que gire el número de grados que sea necesario para que se quede mirando en una determinada dirección. Como es lógico, *n* indica el número asociado con la dirección hacia dónde queremos que mire la tortuga.

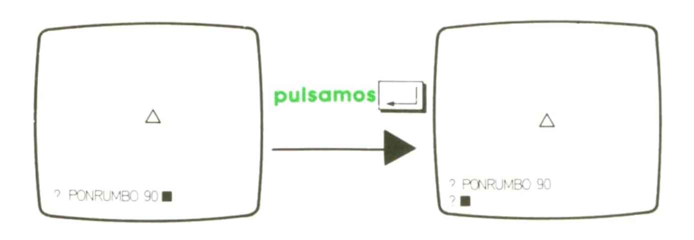
Por tanto, este comando sirve para que la tortuga gire, al igual que GD y GI. Pero hay una diferencia. Con GD y GI se dice que el giro es *relativo*, ya que la tortuga gira el número de grados que nosotros le decimos respecto a la posición que tenga en ese momento. Por el contrario, con PONRUMBO se dice que el giro es *absoluto*, ya que la tortuga se pone a mirar hacia donde nosotros le digamos con independencia de su posición en ese momento, es decir, sin importar hacia donde miraba antes.

Vamos a probar esta diferencia:

— con GD

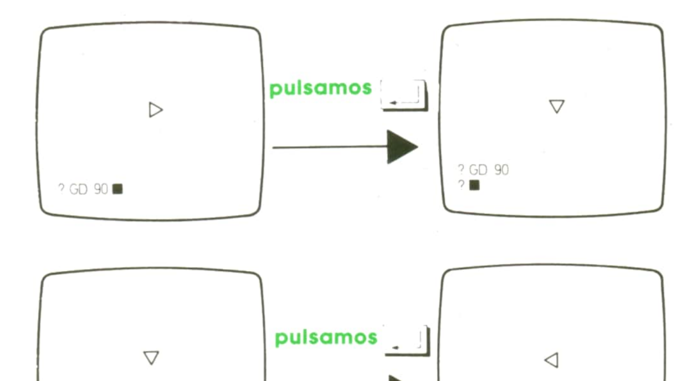


— con PONRUMBO

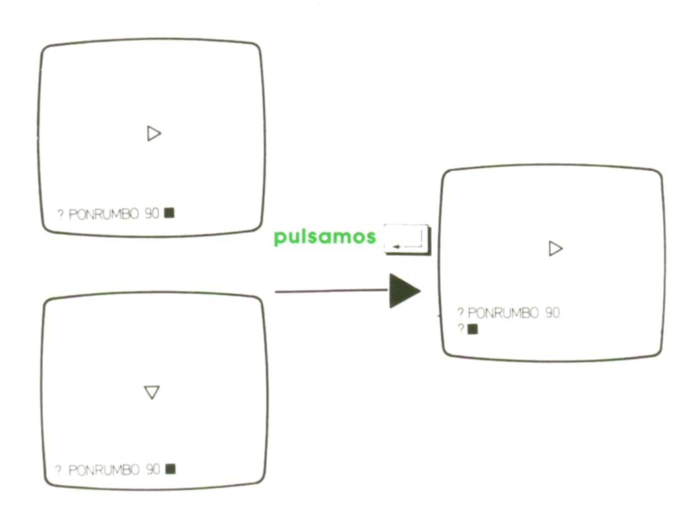


En este caso, ambos tipos de giro han coincidido, pero esto no suele ser lo normal:

— con GD



— con PONRUMBO

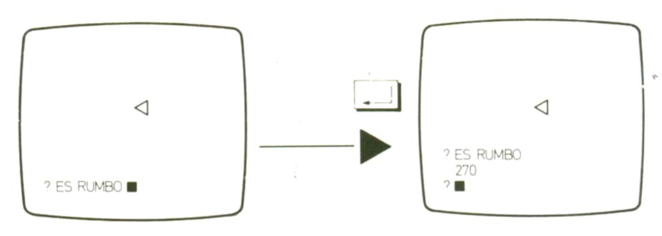


Por otro lado, en un momento determinado podemos desear conocer hacia dónde mira la tortuga porque no lo distinguimos claramente en la pantalla. Para lograrlo, podemos preguntárselo a ella con la función

RUMBO

que nos devuelve un número que indica en qué dirección está la cabeza de la tortuga.

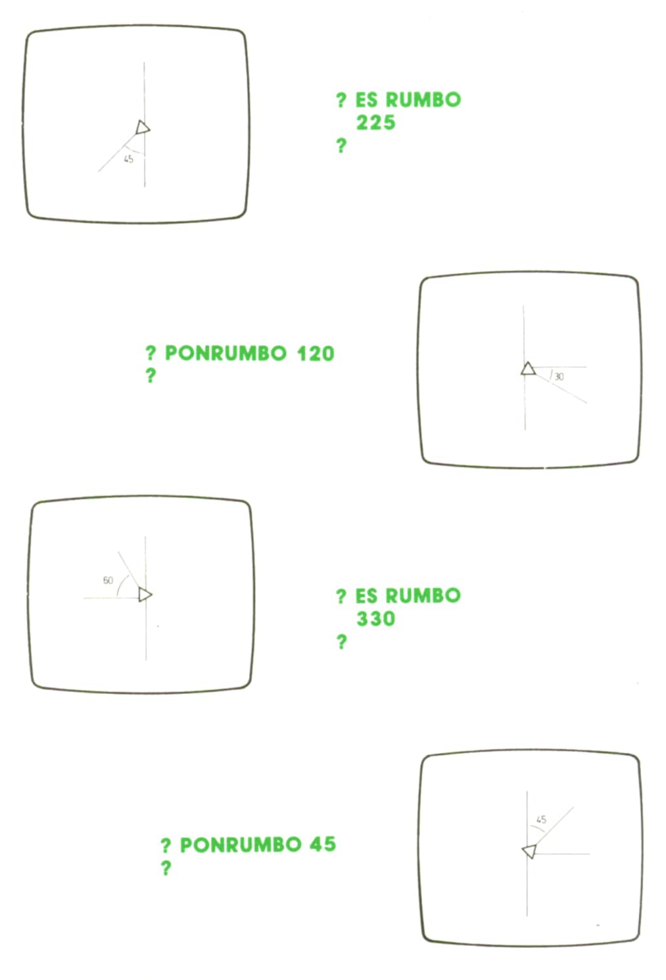
Como siempre, con el resultado de una función hay que hacer algo. En este caso, lo normal es escribirlo. Por ejemplo:



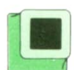
Como era de esperar, la tortuga no ha variado su posición, sólo ha respondido a nuestra pregunta.

Una última cosa: la tortuga no tiene por qué estar siempre mirando a uno de los cuatro puntos cardinales, sino que puede estar orientada hacia una posición intermedia. De la misma forma, en un cierto momento nos puede interesar que la tortuga se oriente hacia una de estas posiciones.

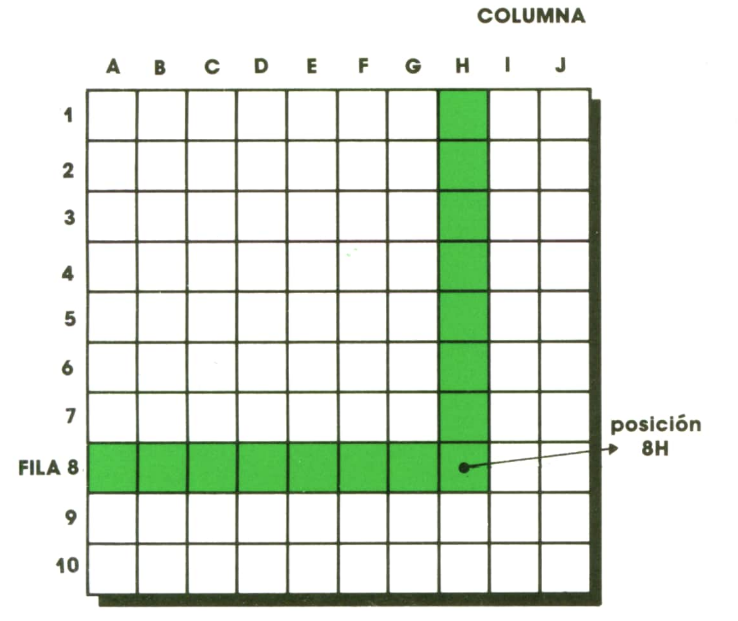
Veámoslo con algunos ejemplos:



Los números que aparecen en las pantallas indican el ángulo que existe entre la posición intermedia y uno de los puntos cardinales.

 La posición de la tortuga

Cuando queremos jugar a los barquitos, cogemos un papel y trazamos una cuadrícula. Para disparar a una determinada posición, la identificamos mediante dos elementos (un número y una letra), que indican la fila y la columna que le corresponden.



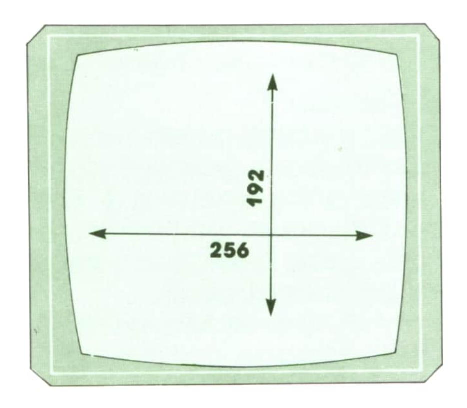
Algo parecido le ocurre a la pantalla por la cual se mueve nuestra tortuga. Aunque nuestro ojo no lo note, en realidad la pantalla es una cuadrícula formada por cuadritos muy pequeños, tan pequeños que nosotros no los distinguimos.

Por tanto, en lugar de decirle a la tortuga que se mueva hacia delante y hacia atrás un número de pasos (con los comandos AV y RE), le podemos mandar que se traslade a una posición determinada (a un cuadrito de la pantalla).

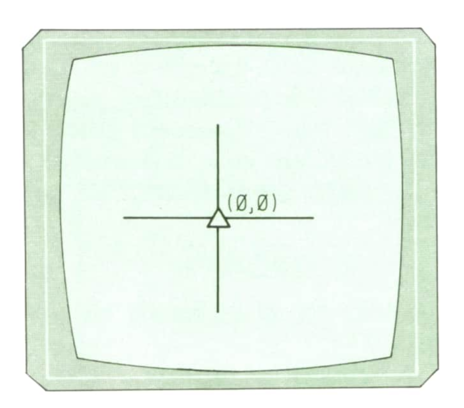
Para indicarle en qué cuadrito queremos que se ponga, vamos a usar dos números (en lugar de un número y una letra). El primer número sirve para dar el valor de la columna de la posición y se llama *coordenada X*. El segundo número da el valor de la fila y se llama *coordenada Y*.

Antes de ver los comandos, nos queda por saber las dimensiones de la pantalla, es decir, los valores mínimos y máximos para las filas y columnas.

Por un lado, la pantalla tiene 192 filas y 256 columnas, pero se numeran partien-



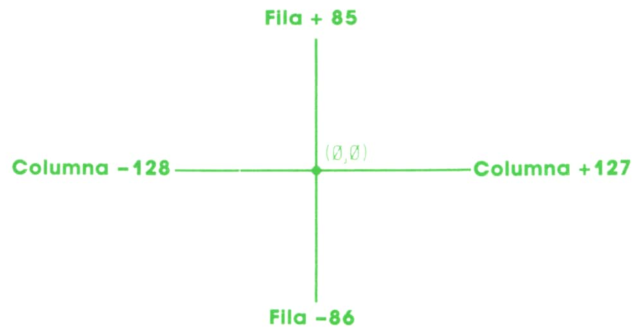
do del centro de la pantalla (posición inicial de la tortuga). Esta posición es la correspondiente a la columna 0 y fila 0.



Para las columnas que están a la derecha de esta posición o en las filas de encima se usan números positivos (números que no llevan nada delante o llevan un signo +).

Para las columnas que están a la izquierda de esta posición o en las filas de debajo se usan números negativos (números que llevan delante un signo -).

Los valores máximos para filas y columnas son:



Es decir, para las coordenadas X e Y se cumple (si la tortuga está en la posición inicial):

— Coordenada X

Sirve para ir a la izquierda (número negativo) o a la derecha (número positivo).

– Coordenada Y

Sirve para ir hacia arriba (número positivo) o hacia abajo (número negativo).

En cuanto a los comandos relacionados con la posición de la tortuga, tenemos de dos tipos: unos sirven para variarla y otros para conocerla.

Si queremos que la tortuga se coloque en una determinada posición de la pantalla usaremos el comando

PONPOS (x y)

donde **x** indica el número de columna e **y** el número de fila de dicha posición.

Si deseamos que cambie de columna pero que siga en la misma fila, tenemos dos opciones. Una de ellas es usar el comando PONPOS poniendo como valor para *x* el que nos interese y para *y* el que tenga la posición de la tortuga en ese momento. Otra posibilidad es utilizar el comando

PONX n

donde **n** indica el número de la nueva columna.

En caso contrario, es decir, si queremos variar de fila pero no de columna, podemos usar PONPOS (al revés que antes) o el comando

PONY n

donde **n** es el número de la nueva fila.

Antes de continuar hay que darse cuenta de dos cosas. En primer lugar, cuando la tortuga cambia de posición no varía su rumbo, es decir, sigue mirando en la misma dirección que antes de moverse. En segundo lugar, cuando la tortuga se desplaza de una posición a otra va dejando un rastro con su lápiz, a no ser que le hayamos dicho antes que lo levante.

Por último, disponemos de unas funciones que nos servirán para que la tortuga nos diga cuál es su posición.

Si queremos saber cuál es la posición de la tortuga usaremos la función

POS

Esta función da como resultado dos números. El primero es el de la columna y el segundo el de la fila.

Si sólo queremos saber el número de columna, la función es:

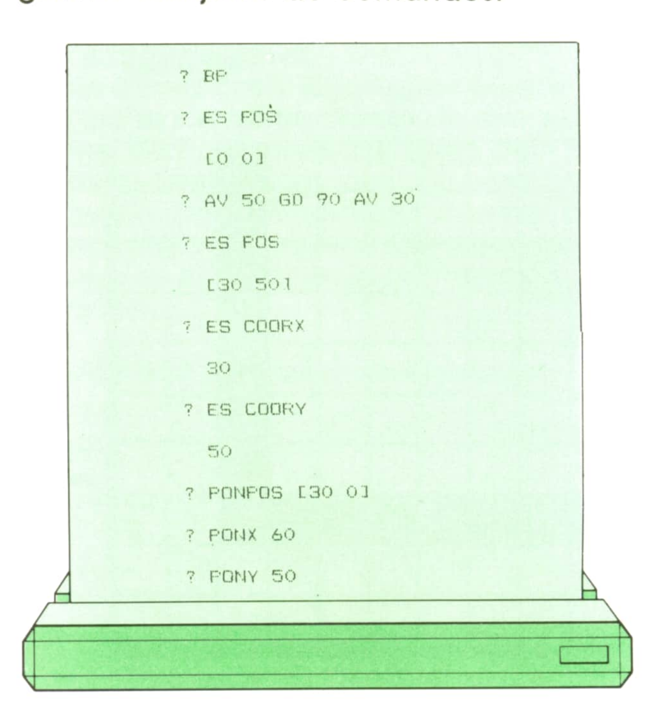
COORX

mientras que si deseamos conocer únicamente el número de fila utilizaremos:

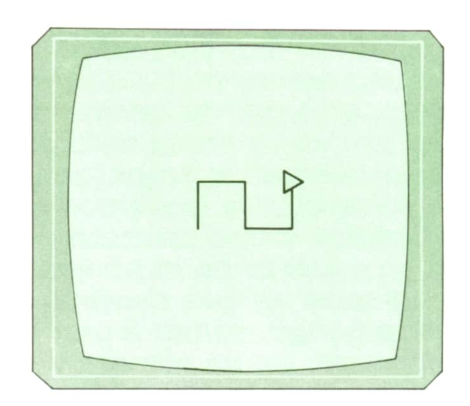
COORY

Como hasta ahora, lo que haremos con estas funciones será escribir su resultado.

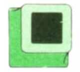
Por ejemplo, podemos probar el siguiente conjunto de comandos:



y nos quedará:

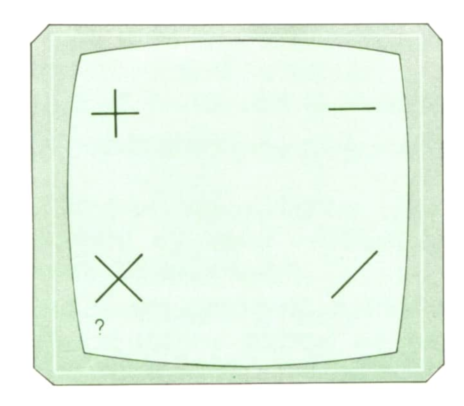


Como vemos, con estos comandos podemos hacer dibujos sin necesidad de preocuparnos del rumbo de la tortuga.



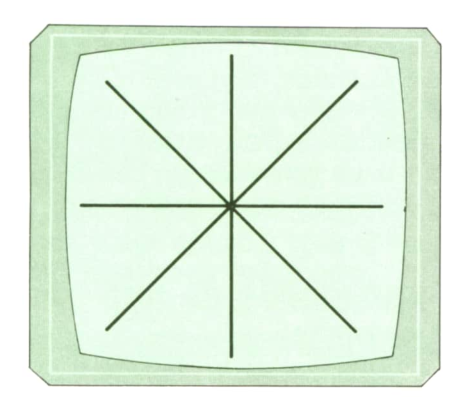
Os proponemos

1. Puedes pintar los signos de las 4 operaciones aritméticas básicas.

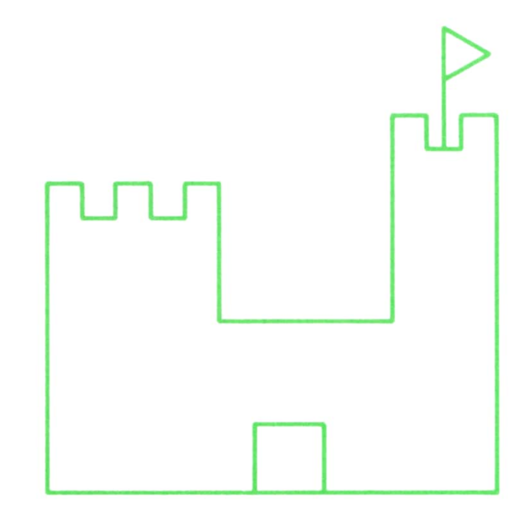


Para moverte a las esquinas utiliza los comandos de posición y para hacer los giros los de rumbo.

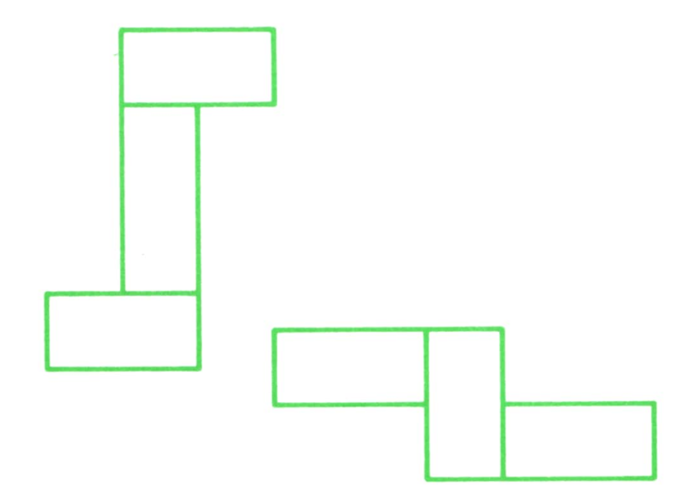
2. Dibujar este sol usando el comando PONRUMBO para que gire la tortuga.



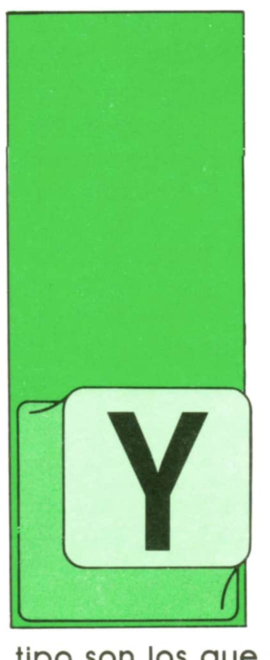
3. Pinta este castillo usando PONRUMBO para girar.



4. Intenta dibujar estas figuras utilizando sólo comandos de posición.



PASCAL



El tipo Boolean (continuación)

Ya hemos aprendido a comparar números entre sí para obtener resultados lógicos; aunque todavía no sabemos cómo, hemos anticipado qué resultados de este tipo son los que se emplean para tomar decisiones.

Además de comparar datos numéricos, es muy frecuente tener que comparar entre sí caracteres; por ejemplo:

"Si el apellido empieza por una letra posterior a la M, entonces..."

Veamos cómo hacer estas comparaciones.

Comparaciones entre caracteres

Dado que los caracteres tienen asociados unos números de orden, y que tenemos la función ORD que nos permite utilizar esos números en expresiones enteras, para ver si un carácter está por detrás o no de la letra M, lo que podríamos hacer, por ejemplo, sería:

ord (Inicial) > ord ('M')

Sin embargo, no es necesario acudir a tal artificio, pues se pueden comparar directamente valores de tipo carácter:

Inicial > **'M'**

Ambas expresiones, a la hora de la verdad, son totalmente equivalentes pero, sin embargo, es mucho más cómodo utilizar la segunda. Los operadores de comparación disponibles son exactamente los mismos que para los datos de tipo INTEGER. Así, 'A' >= 'B' dará el resultado FALSE, pues ord('A') no es mayor o igual que ord('B').

En definitiva, la comparación <, «menor que», se podría llamar ahora «viene antes, por orden alfabético, que» y de manera análoga para las otras.

Como se puede imaginar, esto será muy útil, por ejemplo, a la hora de ordenar alfabéticamente textos de cualquier tipo. Sin embargo, hay que hacer notar que sólo se tiene la seguridad de que están ordenadas correctamente las letras del alfabeto inglés. Con los códigos ASCII empleados en la mayoría de ordenadores personales, por ejemplo, la letra Ñ tiene un número de orden superior a los de las demás letras; además, las letras minúsculas se consideran distintas de las mayúsculas y se encuentran por detrás.

Expresiones de tipo BOOLEAN

De acuerdo con la definición de expresión que hicimos en su momento, una expresión de tipo BOOLEAN es un conjunto de valores de ese mismo tipo combinados entre sí mediante operadores lógicos (o booleanos) para dar a su vez un resultado de tipo BOOLEAN. Los operadores disponibles se representan con unas palabras reservadas, y son los siguientes:

— AND, operación lógica "Y".

La operación AND aplicada a dos valores lógicos da resultado TRUE si y sólo si ambos valores son TRUE simultáneamente:

MayorDeEdad and (Inicial = 'A')

Esta expresión dará resultado TRUE sólo si MayorDeEdad es TRUE y la inicial es la letra A.

— OR, operación lógica "O".

Aplicada a dos valores lógicos resulta TRUE cuando cualquiera de ellos, o los dos a la vez, es TRUE:

(Edad = 29) or (Inicial = 'M')

Para que esta expresión resulte TRUE basta con que Edad sea igual a 29 o que la inicial sea la letra M.

— NOT, operación de negación.

Aplicada a un valor lógico, proporciona su opuesto, es decir, FALSE si era TRUE y viceversa. La expresión:

not (odd(Edad))

resultará TRUE si odd(Edad) es FALSE, es decir, si Edad es par.

— En algunos compiladores de PASCAL existe además la operación XOR u «O exclusivo», similar a OR, pero que resulta TRUE sólo si alguno de los dos valores es TRUE y no si ambos lo son a la vez. Esta operación podría tener un nombre distinto de XOR.

Las expresiones se evalúan de izquierda a derecha, realizándose primero las operaciones NOT, luego las AND y después las demás.

Supongamos que las variables A, B y D tuvieran el valor FALSE y que C tuviera el valor TRUE. Entonces resultaría:

A	or	B	or	C	and	not (D)	primero NOT:
A	or	B	or	C	and	TRUE	luego AND:
A	or	B	or		TRUE		y por último OR:
	FALSE		or		TRUE		
			TRUE				

Al igual que en las expresiones de tipo INTEGER, se pueden utilizar paréntesis para asegurar un determinado orden de evaluación:

(A or B) and (not(C or D) and D)

Hay que hacer notar que, con a los operadores lógicos, existen muchas formas de expresar una condición dada:

(Edad>=30)	equivale a	not (Edad< 30)
(Inicial = 'A')	a	not (Inicial<> 'A')
not (Gordo and Viejo)	a	not (Gordo) or not (Viejo)

(«no ser gordo y viejo a la vez» equivale a decir «no ser gordo o no ser viejo o ninguna de las dos cosas).

Resumiendo en una tabla los resultados de las operaciones lógicas:

A	B	A and B	A or B	A xor B	not A
F	F	F	F	F	T
F	T	F	T	T	T
T	F	F	T	T	F
T	T	T	T	F	F

F = FALSE. T = TRUE.

A y B pueden ser cualquier variable, comparación, etc., que proporcione un resultado de tipo BOOLEAN.

NOTAS:

— Para los expertos en BASIC:

Como se ve, en PASCAL el resultado de una operación lógica sólo puede dar los valores TRUE y FALSE que NO son números. Por tanto, cosas como A = A + (B > 5), tan habituales en BASIC aprovechando que el resultado falso es un 0 y el cierto, generalmente, un −1, no son posibles en PASCAL. NUNCA se pueden mezclar datos de diferente tipo.

— Aunque no resulta de demasiada utilidad, puede ser interesante saber que internamente los valores BOOLEAN, como los CHAR, tienen asociados números ordinales de manera que ORD (TRUE) es mayor que ORD (FALSE) y, por tanto, es posible comparar a su vez valores de tipo BOOLEAN. Ejemplos:

(A = false) dará TRUE si A es igual a FALSE, o sea, equivale a poner NOT (A) y (A = true) equivale a poner A simplemente.

Se puede comprobar que (A <>B) equivale a (A xor B) mirando la tabla anterior.



Toma de decisiones y bucles

Hasta ahora todos los programas no han sido más que simples secuencias de instrucciones que se ejecutaban por orden, empezando por la primera y acabando con la última.

Si se desea construir programas más complejos, es necesario tener la posibilidad de tomar decisiones sobre qué grupos de instrucciones se van a ejecutar en un momento dado y la de repetir bloques de instrucciones.

A continuación describiremos las más importantes estructuras de control existentes en PASCAL.



La estructura IF-THEN

Esta estructura permite decidir durante la ejecución de un programa si una instrucción dada se debe ejecutar o no según una cierta condición. Opcionalmente, es posible escoger entre dos instrucciones. Veamos un ejemplo:

```
program IfThen;

 var N: integer;

begin
 write ('Número: ');
 readln (N);
 if N < 0 then writeln ('¡negativo!');
 writeln (N,' al cuadrado = ', sqr (N));
end.
```

Este programa pide un número y presenta su cuadrado; el mensaje «¡negativo!» sólo aparecerá cuando el número tecleado sea menor que 0. Si traducimos del inglés:

IF	N < 0	THEN	WRITELN ('...
Si	N menor que 0	entonces	WRITELN ('...

vemos claramente cómo es la estructura.

Entre las palabras reservadas IF y THEN se escribe la condición, que puede ser cualquier cosa que dé un resultado de tipo BOOLEAN, o sea, TRUE o FALSE. Durante la ejecución del programa, al llegarse a la estructura IF se evalúa la expresión lógica y, caso de que el resultado sea TRUE, se pasa a ejecutar la instrucción cuya ejecución condicional se desea y que se encuentra tras la palabra THEN. Tras esto se sigue ejecutando lo que hubiera a continuación.

Si el resultado fuese FALSE, se pasaría a ejecutar lo siguiente directamente. En definitiva:

IF (condición) THEN (instrucción a ejecutar en caso de TRUE)

El conjunto así formado es una instrucción en sí misma, aunque al estar formada por varias partes se dice que es una instrucción «ESTRUCTURADA». Como tal instrucción debe separarse de la que pudiera haber a continuación por un punto y coma.

Veamos otro ejemplo:

```
program IfThenElse;

 var N: integer;

begin
 write ('Número: ');
 readln (N);
 if abs (N) > 100  then
    writeln('No me gusta tan grande')
  else
    writeln(N,' al cuadrado = ',
    SQR (N));
 writeln ('Se acabó.')
end.
```

En esta otra variante de la estructura IF, sin embargo, se escoge entre dos posibles instrucciones, según sea el resultado de la condición.

Tras la palabra reservada THEN se escribe aquella que se desea ejecutar cuando el resultado de la condición sea TRUE. A continuación, y separada por la palabra reservada ELSE, se escribe la que hay que ejecutar en caso de ser FALSE el resultado.

Sea cual sea el resultado de la condición, y tras ejecutarse la instrucción correspondiente, se continúa con la siguiente a la estructura.

Una vez más, la traducción del inglés es clara:

IF	abs(N) > 100	THEN	WRITELN ('No...
Si	valor absoluto de N mayor que 100	entonces	WRITELN ('No...
	ELSE	WRITELN (N,' al...	
	en otro caso	WRITELN (N,' al..."	

Nótese que tras el primer WRITELN y antes de ELSE no hay ningún punto y coma; si lo hubiera, al llegar a él el compilador, supondría que es el final de una estructura IF del primer tipo, con lo que al llegarse a la palabra ELSE se produciría un error.

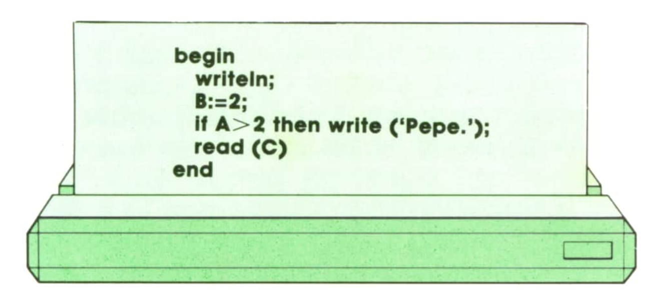
El conjunto formado por la condición, las dos instrucciones y las palabras reservadas IF, THEN y ELSE es a su vez una instrucción (estructurada, eso sí), y, por tanto, debe ir separada de la siguiente por un punto y coma.

La instrucción (o instrucciones) cuya ejecución condicional se desea puede ser cualquiera que se nos ocurra, simple o estructurada. Entre estas últimas se encuentra lo que se denomina SECUENCIA DE INSTRUCCIONES.



Secuencia de instrucciones

Una SECUENCIA o BLOQUE DE INSTRUCCIONES es un conjunto de instrucciones de cualquier tipo escritas una detrás de otra, separadas entre sí por punto y coma, y enmarcadas por las palabras reservadas BEGIN para indicar el comienzo, y END para indicar el final. Por ejemplo:



Toda la secuencia es como si fuese una única instrucción formada por varios pasos. Como con otras instrucciones estructuradas, el PASCAL es muy claro respecto a su uso: se puede poner en cualquier sitio en que pudiera figurar una instrucción simple, y su ejecución consiste en ejecutar por orden las instrucciones que la integran.

Utilizando secuencias en una estructura IF-THEN es posible, por tanto, decidir si se ejecutan o no grupos complejos de instrucciones. Por ejemplo:

```
program Secuencia;

const
  EdadMaxima = 15;
var
  Inicial: char;
  Peso   : integer;

begin
 writeln ('Inicial? ');  readln (Inicial);

 (* Miramos a ver si se ha tecleado una letra: *)
 if (Inicial < 'A') or (Inicial > 'Z') then
    begin
      writeln ('Imposible.');
      writeln ('Supongo que es la A');
      Inicial:= 'A'
    end;     (* Aquí acaba un conjunto IF-THEN *)

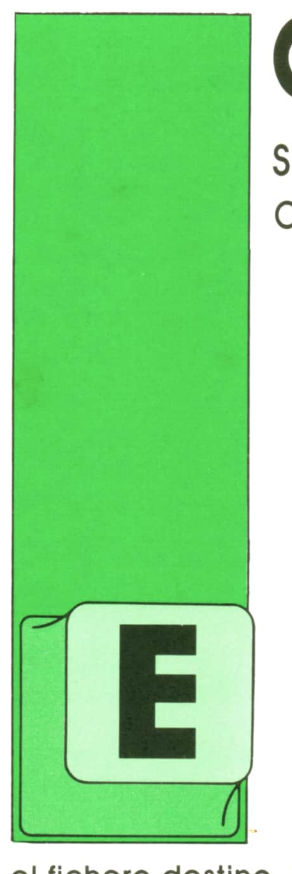
 writeln ('Peso? ');  readln (Peso);

 if  Peso > 100 then
      begin
        writeln ('Pues no está mal.');
        writeln (Peso,'Kg...¡caray!')
      end
   else
      if Peso < 30 then
           writeln ('Muy poco, ¿no?');
           (* Aquí acaba el último IF-THEN *)
  (* Aquí acaba el conjunto IF-THEN-ELSE *)

 writeln('Letra= ',Inicial,'  Peso= ',Peso)
end.
```

(Nótese la indentación utilizada para hacer el programa más claro.)

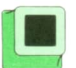
Como la estructura IF-THEN en su conjunto es una instrucción estructurada, podría formar parte de una secuencia o ser incluso una de las instrucciones que forman parte de otra estructura IF, que es lo que sucede en el programa: si Peso es mayor que 100, saca dos mensajes y en otro caso... si Peso es menor que 30, saca otro mensaje.



OTROS LENGUAJES

SISTEMAS OPERATIVOS MS-DOS (III)

COMANDOS PARA OPERACIONES CON FICHEROS

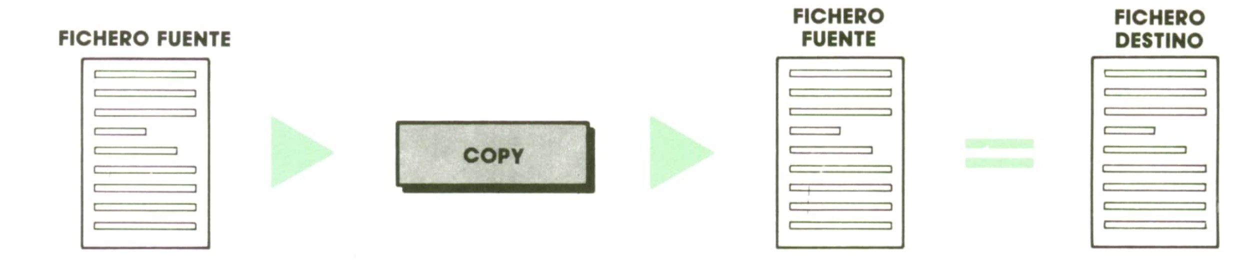


COPY

EL comando COPY permite realizar copias de uno o más ficheros a un dispositivo de almacenamiento específico. Primero se debe indicar el fichero origen, y después el fichero destino. Por ejemplo:

COPY ARCHIVO1 ARCHIVO2

haría que en el fichero "ARCHIVO2" se copie íntegramente la información del fichero "ARCHIVO1".



Una de las funciones para la que más se utiliza COPY es para realizar «reproducciones» o copias de ficheros con el mismo o distinto nombre, y en el mismo o diferente dispositivo de almacenamiento.

También se puede copiar uno o más ficheros en un directorio especificado. Para ello, el segundo parámetro del comando debe ser el nombre del directorio destino. Si detrás de éste no hay ningún nombre de archivo, los ficheros copiados toman el mismo nombre que los originales.

Otra función que se puede realizar con el comando COPY es la concatenación o unión de ficheros, para lo cual se utiliza como operador el signo "+". Por ejemplo:

COPY FICH1 + FICH2 FICH3

"une" los ficheros "FICH1" y "FICH2" y los copia en el archivo "FICH3".

Además de las diversas opciones de trabajo, el comando COPY tiene varios indicadores, que sirven para verificar el proceso de copia, y especificar el tipo de contenido de los ficheros copiados, ya sea código ASCII o código binario.

ERASE

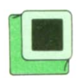
Con el comando ERASE (en algunas versiones de MS/DOS se llama DELETE) se borra un fichero, o un grupo de ficheros en caso de utilizar caracteres comodín.

Dado que esta orden no pide confirmación para realizar la acción de borrado, es conveniente que nos aseguremos que la especificación del archivo (o archivos) a borrar es correcta.

RENAME

Esta orden permite cambiar el nombre a los ficheros ya creados. La única limita-

ción estriba en la imposibilidad de cambiar el nombre a un grupo de ficheros (proceso que se realiza utilizando los caracteres comodín), en el que no estén todos almacenados en el mismo disco.



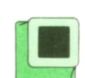
Comandos de salida de ficheros

TYPE

Este comando visualiza por pantalla el contenido de un fichero especificado, sin tener posibilidad de modificarlo. El proceso de salida de la información se realiza de forma continua, de manera que si el tamaño del fichero excede a la capacidad visual de la pantalla (24 líneas normalmente), la información se desplaza hacia arriba para dar cabida a nuevas líneas. En caso de que se quiera parar este desplazamiento es necesario una combinación de teclas específicas. El texto vuelve a desplazarse pulsando cualquier tecla.

PRINT

Con el comando PRINT se pueden imprimir uno o varios ficheros mientras MS/DOS realiza otras tareas. Esto se consigue utilizando los tiempos muertos de la UCP, de manera que cuando no hace otros trabajos se ocupa de ir imprimiendo los ficheros especificados.



Comandos de manejo de la estructura de ficheros

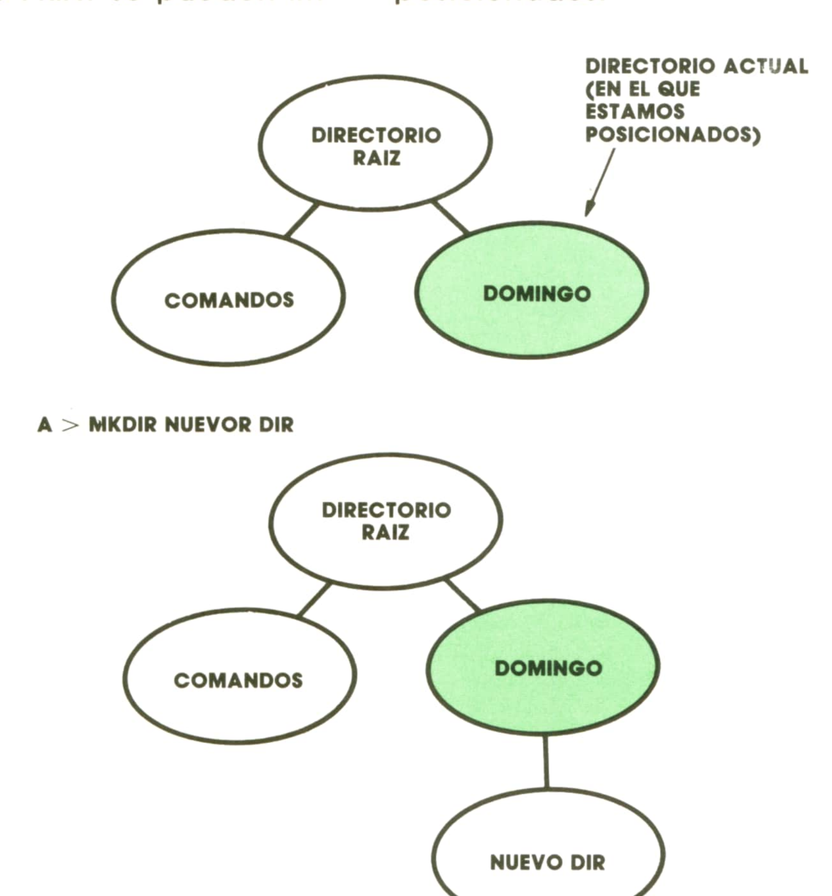
Dentro de este grupo hemos encuadrado aquellos comandos que nos permiten crear y borrar directorios, así como movernos a través de la estructura en árbol de los ficheros.

MKDIR

Con esta orden podemos crear un subdirectorio en un disco especificado. Por ejemplo, con la orden:

MKDIR NuevoDir

se crearía un subdirectorio al que se le da el nombre "NuevoDir", y que está "colgado" del directorio donde estamos posicionados.




Ejemplo de la ejecución del comando MKDIR.

RMDIR

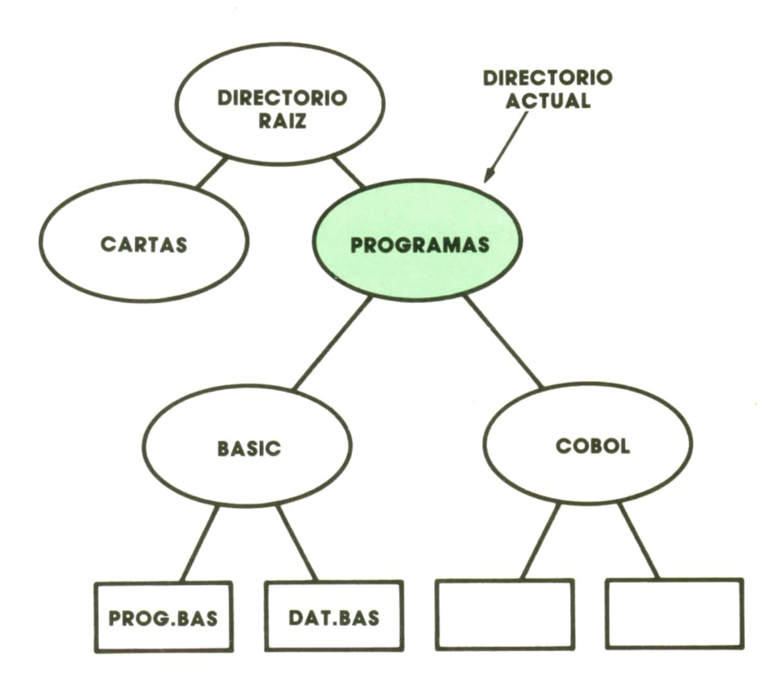
Con el comando RMDIR borramos del disco especificado el directorio (o subdirectorio) nombrado, el cual debe estar vacío previamente.

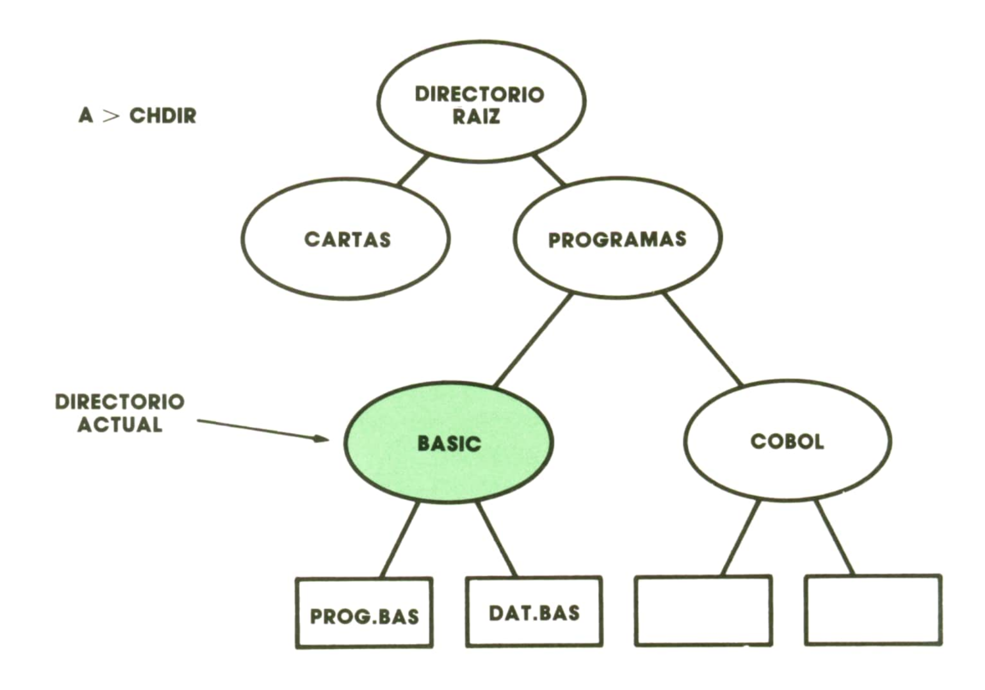
No puede eliminarse el directorio raíz ni el directorio actual (aquél en el cual estamos).

CHDIR

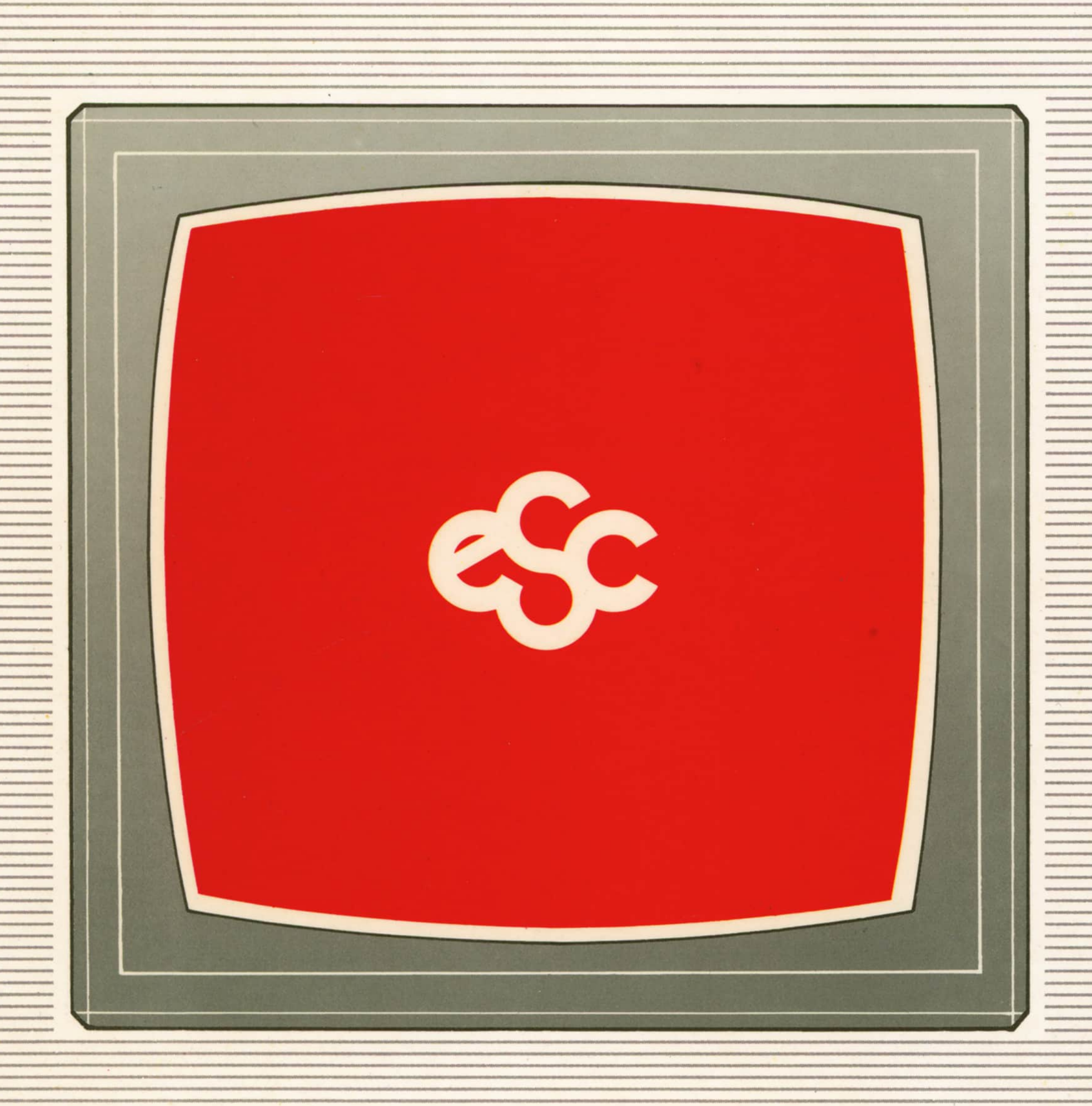
Esta orden cambia el directorio actual de la unidad de disco especificada, o bien visualiza el camino desde el directorio raíz hasta el directorio actual.

Si después de CHDIR va un nombre de subdirectorio, se "baja" al subdirectorio indicado. En caso de que se quiera "subir" al directorio "padre" se teclea CHDIR.. (los dos puntos especifican el directorio padre).





Cambio de directorio actual mediante el comando MKDIR.



▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼